MockCert

Study guide | Databricks | Generative AI Engineer Associate

Databricks Certified Generative AI Engineer Associate

Notes for every domain of the Databricks Certified Generative AI Engineer Associate exam, organized by the official exam guide's domains and weights.

By MockCert about 159 min read Updated November 17, 2026

CONTENTS

DOMAIN 1 · 14%
Design Applications 2

DOMAIN 2 · 14%
Data Preparation 15

DOMAIN 3 · 30%
Application Development 32

DOMAIN 4 · 22%
Assembling and Deploying Applications 60

DOMAIN 5 · 8%
Governance 92

DOMAIN 6 · 12%
Evaluation and Monitoring 101

MockCert is an independent certification preparation platform. It is not affiliated with, endorsed by or operated by Databricks. Exam domains and weights follow the official exam guide. These notes were last updated on November 17, 2026; the notes at mockcert.com are kept up to date as the exam changes.

Domain 1 | 14% of the exam | 17 min read

Design Applications

Overview

This domain is about 14% of the exam, roughly 6 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Design a prompt that elicits a specifically formatted response

Start here

THINK FIRST

How would you get a model to answer in exactly the format you need, and prove that it does?

Why it matters: Questions describe an output format a team needs and ask how to template, version or test the prompt behind it.

A well-formatted answer starts with a clear prompt template and ends with a test. The Prompt Registry stores templates as immutable versions, and evaluations run every version on the same dataset and judges.

Key terms

Term	Meaning
Prompt Registry	MLflow's central store for managing prompt templates across their lifecycle.
Prompt version	An immutable snapshot of a prompt with an auto-incrementing number.
Alias	A mutable pointer that refers to one specific prompt version.
Judge	An evaluator, such as one built with `make_judge`, that checks an output against a rule.

Two prompt shapes

Pick Text for completion prompts and Chat for conversations. A Text prompt is one template. A Chat prompt is a list of role-based messages, such as system and user, for conversational models. In a conversation template, each message is a dict with `role` and `content` keys.

Text and Chat prompts compared

Feature	Text	Chat
Shape	A single template	A list of role-based messages

Feature	Text	Chat
Best for	Completion-style prompts	Conversational models

Template variables

Registry templates mark variables with double braces. Write variables as `{{name}}` in both simple and conversation prompts. LangChain and LlamaIndex expect single braces, so MLflow can convert a registered prompt to that format for them.

Example: A registry template with one variable in double braces.

```
"Hello {{name}}"
```

- `{{name}}`: the variable filled in at run time

EXAM TRAP

Single-brace templates are not the registry's native syntax. They are the converted form for libraries like LangChain.

Spell out the format

Show the model the exact output you expect. Prompt optimization uses data with expected responses to steer the output. For example, you can optimize for one word chosen from **5** options, with no explanation added.

Versions and aliases

A saved prompt version can never be edited. To change a prompt, register it again under the same name with `mlflow.genai.register_prompt()`. That adds a new version, keeps the full history and lets you roll back. Names allow only letters, numbers, hyphens, underscores and dots.

EXAM TRAP

Registering an existing prompt name does not overwrite it or fail. It creates the next version of that prompt.

Check format with a judge

A custom judge can test format compliance. A judge built with `make_judge` can check, say, that a summary has exactly **2** sentences. Use `feedback_value_type=bool` for pass/fail judges, because MLflow averages the results into a pass rate.

Compare versions fairly

Evaluate every version on the same data and judges. The Compare button only shows a text diff between templates. To learn which version produces better output, run evaluations on each and compare the evaluation runs.

EXAM TRAP

A text diff does not tell you which prompt performs better. To compare outputs, run evaluations on both versions with the same dataset and judges.

Permissions you need

Viewing or creating prompts needs a prepared schema. The Unity Catalog schema must grant CREATE FUNCTION, EXECUTE and MANAGE before you can view or create prompts in the registry.

Key takeaways

- Prompt versions are immutable; editing means registering a new version.
- Registry templates use double braces, like `{{name}}`, for variables.
- Use a bool judge to turn format checks into a pass rate.
- Compare prompt versions with evaluations on the same data, not with a text diff.

Sources: Prompt Registry | Databricks on AWS (opens in a new tab) · Evaluate and compare prompt versions | Databricks on AWS (opens in a new tab) · Create and edit prompts | Databricks on AWS (opens in a new tab) · Optimize prompts tutorial | Databricks on AWS (opens in a new tab)

Select model tasks to accomplish a given business requirement

Start here

THINK FIRST

How do you match a business need to the right AI task and Databricks function?

Why it matters: Questions describe a business requirement and ask which model task or AI Function fits it.

Start by naming the task the business wants, such as summarizing, translating or classifying. Purpose-built functions such as `ai_summarize`, `ai_translate` and `ai_classify` cover those tasks. `ai_query` is the general-purpose option for task and model flexibility.

Key terms

Term	Meaning
Task-specific AI Function	A purpose-built function tuned for one job, such as classification or sentiment analysis.
ai_query	The general-purpose function: you supply a prompt and choose a supported foundation model.
Summarization	The NLP task category for condensing text into a short gist.
Provisioned throughput	A Foundation Model APIs mode for production workloads that need high throughput or guarantees.

Name the task first

The business requirement tells you the NLP task category. A request to condense a paragraph into a one-sentence gist is summarization. So you evaluate candidate models under the Summarization category.

EXAM TRAP

A one-sentence gist is not classification or entity extraction. Condensing text is summarization, so compare models on that task.

Map needs to AI Functions

Task-specific functions each target one task, such as classification. Databricks suggests starting with these functions. They use research techniques Databricks maintains, and they need no customization.

Business needs matched to task-specific AI Functions

Business need	Function	What it does
Shorten long text	`ai_summarize`	Generates a summary of text
Reach readers in another language	`ai_translate`	Translates text to a chosen target language
Sort text into categories	`ai_classify`	Classifies text by labels you supply
Gauge customer mood	`ai_analyze_sentiment`	Runs sentiment analysis on text
Hide sensitive entities	`ai_mask`	Masks the entities you specify in text
Compare two pieces of text	`ai_similarity`	Scores how similar two strings are in meaning
Predict future values	`ai_forecast`	Extrapolates time series data to a horizon

EXAM TRAP

`ai_forecast` is not a text function. It is a table-valued function that extends time series into the future.

The general-purpose option

`ai_query` **is the general-purpose function for task and model flexibility.** You write the prompt and choose any supported Foundation Model API.

EXAM TRAP

For batch work with `ai_query`, do not reach for provisioned throughput. Databricks-hosted models prefixed `databricks-` work best for batch.

Picking a serving mode

Provisioned throughput suits demanding production workloads. Choose it for high throughput, performance guarantees, fine-tuned models or extra security needs. A foundation model also suits a quick proof-of-concept, or a RAG chatbot with a vector index.

Where AI Functions run

AI Functions need Databricks Runtime 15.4 LTS or above. They are not available on Classic SQL warehouses. Check the compute before you plan a solution around them.

Key takeaways

- Condensing text into a short gist is a summarization task.
- Start with a task-specific AI Function; use `ai_query` when you need any prompt or model.
- AI Functions need Runtime **15.4 LTS** or above and do not run on Classic SQL warehouses.
- Batch jobs with `ai_query` work best on Databricks-hosted `databricks-` models.

Sources: Databricks Exam Guide Databricks Certified Generative AI Engineer Associate (opens in a new tab) · Transform unstructured data using AI Functions | Databricks on AWS (opens in a new tab) · Databricks Foundation Model APIs | Databricks on AWS (opens in a new tab)

Select chain components for a desired model input and output

Start here

THINK FIRST

Given what a chain must take in and hand back, which components and model settings would you put in it?

Why it matters: Questions describe a chain's input and required output, and ask which component, model or setting fits.

A RAG chain is built from components that each do one job. Query understanding clarifies intent to improve retrieval, prompt augmentation joins the query with retrieved information, and post-processing keeps the LLM's responses on-topic and within constraints.

Key terms

Term	Meaning
Query understanding	Transforming the user's query to capture intent and pull out filters or keywords.
Prompt augmentation	Joining the user query with retrieved information and instructions for the LLM.
Guardrails	Safety checks that keep responses on-topic, factually consistent and within constraints.

Term	Meaning
Function calling	The LLM returns a JSON object naming functions; your code makes the actual calls.

Right-size the models

Different chain steps can use different models. An expensive general-purpose model may be overkill for a step like detecting a query's intent. In LangChain, the `ChatDatabricks` class queries a served model.

Example: Creates a LangChain chat model object for a model served on Databricks.

```
my_llm = ChatDatabricks(model="my_model")
```

Structured output

Request structured output through the `response_format` field. It can return text, free-form JSON or JSON that follows a schema. The request format stays the same whatever the model provider. A schema holds at most **64** keys.

EXAM TRAP

For Anthropic Claude models, `json_object` is not supported. Use `json_schema`, or leave out `response_format` for unconstrained output.

Tool calls

With function calling, the model writes JSON and your code runs the function. The default `tool_choice` is `"auto"`, so the model decides whether and which functions to call. You can define at most **32** functions, and parallel function calling is not supported.

EXAM TRAP

The LLM never runs a function itself. It only returns a JSON object that your code uses to make the call.

Guard the output

Add a post-processing step to validate and secure responses. Instructor or Outlines are good starting points for validating structured output. Llama Guard, or a moderation API such as OpenAI's, can provide safety guardrails.

Key takeaways

- Query understanding refines the user's query, prompt augmentation builds the prompt, and guardrails check the response.
- A costly general-purpose model can be overkill for simple steps like intent detection.
- Function calling returns JSON for your code to act on, with up to **32** functions.
- Validate structured output with Instructor or Outlines; screen safety with Llama Guard.

Sources: Improve RAG chain quality | Databricks on AWS (opens in a new tab) · Function calling on Databricks | Databricks on AWS (opens in a new tab) · Query a chat model | Databricks on AWS (opens in a new tab) · Structured outputs on Databricks | Databricks on AWS (opens in a new tab)

Translate business use case goals into a description of the desired inputs and outputs for the AI pipeline

Start here

THINK FIRST

How do you turn a business goal into a clear spec of what goes into an AI pipeline and what must come out?

Why it matters: Questions describe a business goal and ask which inputs, output format or pipeline steps meet it.

Start from the goal, then name the inputs the pipeline needs and the shape of its outputs. A goal like sorting reviews by sentiment implies documents going in and labeled, structured records coming out.

Key terms

Term	Meaning
Structured outputs	Responses generated in a defined JSON format, requested through a `response_format` field.
JSON schema	The definition a structured response must follow; it may hold at most **64** keys.
VARIANT	The structured type `ai_parse_document` returns from raw PDF bytes, with elements, layout and file details.
Batch inference pipeline	An AI Functions workflow that covers ingestion, preprocessing, inference and post-processing.

Inputs follow the goal

The data a pipeline needs depends on the app's requirements and goals. For a customer support bot, inputs might be knowledge base articles, FAQs, product manuals and specifications, and troubleshooting guides. Domain experts and stakeholders help, because they know what queries users are likely to send.

Choose the output shape

Output can be text, free-form JSON or JSON that follows a schema. Structured outputs fit extraction from large volumes of documents. One example is labeling product reviews as negative, positive or neutral. Any supported chat model can use them via the `response_format` field.

EXAM TRAP

A goal that needs the same fields every time, such as a sentiment label per review, points to structured outputs, not free text.

Design the schema well

Keep schemas flat and within 64 keys. Heavily nested schemas lower generation quality, so flatten them when you can. Foundation Model APIs do not support `pattern` or composition keywords such as `anyOf`, `oneOf`, `allOf`, `prefixItems` or `$ref`.

EXAM TRAP

Validating a field with a regular expression through `pattern` is not supported in Foundation Model APIs structured outputs.

Where structured outputs work

Pay-per-token and provisioned throughput chat endpoints support them. For Anthropic Claude models, only `json_schema` is allowed, not `json_object`. Streaming is not supported there either, so set `stream` to false whenever you pass `response_format`.

EXAM TRAP

For Claude models, `json_object` is not an option. Use `json_schema`, or leave out `response_format` for unconstrained output.

Map the pipeline steps

Batch AI Functions pipelines run from ingestion to post-processing. In a document pipeline, `ai_parse_document` turns raw PDF bytes into a structured VARIANT. `ai_classify` accepts that VARIANT directly, with no cast to string. Classic SQL warehouses cannot run AI Functions.

Key takeaways

- Let the app's goals decide which documents and data feed the pipeline.
- Use structured outputs when a goal needs consistent fields from many documents.
- Keep JSON schemas flat, at most **64** keys, and free of `pattern` and `anyOf`.
- `ai_classify` takes the VARIANT from `ai_parse_document` directly.

Sources: Structured outputs on Databricks | Databricks on AWS (opens in a new tab) · Build an unstructured data pipeline for RAG | Databricks on AWS (opens in a new tab) · Document Intelligence powered by AI Functions | Databricks on AWS (opens in a new tab) · Deploy batch inference pipelines | Databricks on AWS (opens in a new tab)

Define and order tools that gather knowledge or take actions for multi-stage reasoning

Start here

THINK FIRST

How would you choose and describe the tools an agent calls across several reasoning steps?

Why it matters: Questions describe an agent's data and task and ask which tool type, server or setting fits it.

A SQL function fits a fixed query, and the Databricks SQL MCP fits agent-written SQL. An AI Search index serves documents, and Files API tools read volume files. Clear descriptions and comments tell the agent when to call each tool.

Key terms

Term	Meaning
Unity Catalog function tool	A SQL function registered in Unity Catalog that an agent calls, passing its own parameter values.
MCP server	Databricks SQL, AI Search and Genie One are MCP servers; Databricks-provided and registered ones follow Unity Catalog permissions.
VectorSearchRetrieverTool	The class that turns an AI Search index into a tool for a Python agent.
Supervisor Agent	A legacy offering, not recommended for new systems, that coordinates up to **50** agents and tools.

Match the tool to the query

Known query, unknown values: use a Unity Catalog SQL function. The query shape is fixed ahead of time, and the agent only fills in the parameters. When the agent must compose its own SQL against Unity Catalog tables, the Databricks SQL MCP fits instead.

Three ways an agent can reach structured data

Feature	UC SQL function	Databricks SQL MCP	Genie One MCP
Best fit	A query known ahead of time	Specific SQL queries and data engineering	Business questions in natural language
How it answers	Runs the fixed query with the agent's parameters	The agent writes and runs its own SQL	Grounds answers in Genie Ontology

EXAM TRAP

A SQL function is the wrong pick when the agent must write new queries. It only fits a query fixed in advance.

Search unstructured documents

For new Python agents, AI Bridge turns an AI Search index into a tool. Expose the index with `VectorSearchRetrieverTool`, available in `databricks-langchain` and `databricks-openai`. Direct-access indexes, and Delta Sync indexes with self-managed embeddings, also need a custom embedding model and text column configured.

EXAM TRAP

The AI Search MCP server does not work with every index. It needs an index that uses Databricks managed embeddings.

Retriever output shape

A retriever tool returns a `List[Document]`. That shape matches the MLflow retriever schema. Extra attributes such as `doc_uri` and `similarity_score` go in each Document's `metadata` field.

Files and Genie as tools

Files API tools read volumes; AI Bridge can wrap a Genie Agent. For reports, text documents or configuration files in a Unity Catalog volume, build tools on the Databricks SDK Files API. The agent's identity needs `USE CATALOG`, `USE SCHEMA` and `READ VOLUME`. AI Bridge can expose a specific Genie Agent, handling requests, polling and results.

EXAM TRAP

Granting only `READ VOLUME` is not enough. The agent's identity also needs `USE CATALOG` and `USE SCHEMA` to reach the volume.

Describe tools so the agent chooses well

The agent reads a tool's description to decide when to call it. When a Unity Catalog function wraps an AI Search query, its comment is that description. Say what kinds of documents and information the index holds.

Coordinating many tools

Supervisor Agent coordinates up to 50 agents and tools, but it is legacy. It delegates using each tool or subagent description, and includes a code execution tool by default. For new low-code agents, pick Genie Agents; to orchestrate several agents in code, the Agent Bricks CLI fits.

EXAM TRAP

Supervisor Agent is not the recommended choice for a new system. It is a legacy offering.

Governing MCP tools

Databricks-provided and registered MCP servers follow Unity Catalog permissions. Unity Gateway controls access, manages credentials and monitors usage. Three Beta tools, `system.ai.dbsql`,

`system.ai.web_search` and `system.ai.sandbox`, need an account admin to enable the Enhanced Unity Gateway preview.

Key takeaways

- Fixed query with agent-supplied values: Unity Catalog SQL function. Agent-written SQL: Databricks SQL MCP.
- Retriever tools return `List[Document]`, with extras like `doc_uri` in the `metadata` field.
- Descriptions and function comments are how an agent decides which tool to call.
- Supervisor Agent handles up to **50** agents and tools but is legacy; prefer Genie Agents or the Agent Bricks CLI.

Sources: Connect agents to structured data | Databricks on AWS (opens in a new tab) · Connect agents to unstructured data | Databricks on AWS (opens in a new tab) · Use Supervisor Agent to create a coordinated multi-agent system (legacy) | Databricks on AWS (opens in a new tab) · MCPs | Databricks on AWS (opens in a new tab)

Determine how and when to use Agent Bricks (Knowledge Assistant, Multiagent Supervisor, Information Extraction) to solve problems

Start here

THINK FIRST

When would you reach for Knowledge Assistant, Supervisor Agent or Information Extraction to solve a problem?

Why it matters: Questions describe a business problem and ask which Agent Bricks agent fits it and how to set it up.

Each Agent Bricks agent solves one kind of problem. Knowledge Assistant answers questions over documents, Information Extraction turns text into structured data, and Supervisor Agent coordinates many agents. Knowledge Assistant and Supervisor Agent are now legacy.

Key terms

Term	Meaning
Knowledge Assistant	A chatbot that answers questions about your documents, with citations.
Information Extraction	An agent that turns unstructured documents and text into structured insights using a schema.
Supervisor Agent	An agent that coordinates Genie Agents, endpoints, functions, MCP servers and custom agents.
Instructed Retriever	The approach Knowledge Assistant uses to address the limits of traditional RAG.

Pick the agent by problem

Match the agent to the shape of the problem. Ask whether users need answers over documents, structured fields pulled from text, or work spanning several specialized domains.

The three Agent Bricks agents compared

Feature	Knowledge Assistant	Information Extraction	Supervisor Agent
Solves	Questions about your documents, answered with citations	Turning unstructured text into structured insights	Complex tasks across specialized domains
Key input	Volume files or AI Search indexes	A defined schema	Up to **50** agents

EXAM TRAP

Knowledge Assistant and Supervisor Agent are legacy. For new low-code agents, pick Genie Agents; for agents built in code, use the Agent Bricks CLI.

Knowledge Assistant inputs

Volume files can be txt, pdf, md, ppt/pptx or doc/docx. Ingestion skips any file over **100 MB**. One assistant draws on at most **10** knowledge inputs. AI Search indexes must use `databricks-qwen3-embedding-0-6b`, `databricks-gte-large-en` or `databricks-bge-large-en` embeddings.

EXAM TRAP

Adding files does not update the assistant by itself. You must click Sync, and syncing runs incrementally.

Improving answers with experts

Subject matter experts tune Knowledge Assistant with natural language feedback. They describe what should change, and the assistant adjusts its quality and behavior to match.

Extraction at scale

Run Information Extraction on new data with a scheduled Lakeflow pipeline. The pipeline invokes the agent at set intervals and writes the extracted data into a streaming table.

Supervisor limits and access

One supervisor system can use at most 50 agents. Built-in access controls let end users reach only the subagents and data they may access. To call a Knowledge Assistant endpoint as a subagent, users need CAN QUERY on it. Every supervisor also has a code execution tool for Python, SQL and shell.

EXAM TRAP

A supervisor does not bypass permissions. Without explicit access to a subagent, it cannot return helpful answers from that subagent.

Key takeaways

- Documents to chatbot: Knowledge Assistant. Text to structured data: Information Extraction. Many agents together: Supervisor Agent.
- Knowledge Assistant skips files over **100 MB**; click Sync after adding files.
- A supervisor system holds at most **50** agents and respects each user's own access.
- For new systems, use Genie Agents or the Agent Bricks CLI instead of the legacy agents.

Sources: Use Knowledge Assistant to create a high-quality chatbot over your documents | Databricks on AWS (opens in a new tab) · Information Extraction | Databricks on AWS (opens in a new tab) · Use Supervisor Agent to create a coordinated multi-agent system (legacy) | Databricks on AWS (opens in a new tab)

Domain 2 | 14% of the exam | 23 min read

Data Preparation

Overview

This domain is about 14% of the exam, roughly 6 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Apply a chunking strategy for a given document structure and model constraints

Start here

THINK FIRST

How do a document's layout and your model's limits shape the chunk size, overlap and tools you choose?

Why it matters: Questions describe documents and model constraints and ask which chunking, parsing or embedding choice fits.

Chunking is a trade-off. Chunk size and overlap set how many embeddings you create, while model context and service limits cap what each step can take. Parse complex documents first, then chunk to fit the model.

Key terms

Term	Meaning
Chunk overlap	Text shared between neighboring chunks; more overlap means more chunks.
ai_parse_document	The Document Parsing function that pulls structured data out of files like PDFs and slides.
ai_prep_search	A Beta step that prepares parsed output for retrieval.
Context length	How many tokens an embedding model accepts per input.

Size and overlap set the count

Bigger chunks and less overlap mean fewer embeddings. Smaller chunks or more overlap raise the chunk count, and with it the number of embeddings. When you must stay under an embedding limit, grow the chunks or cut the overlap.

EXAM TRAP

Adding overlap does not reduce the number of chunks; it increases the embedding count.

Match the model to the chunk

Fit the embedding model's context to your chunk cap. If chunks are capped at **512** tokens and cost and latency matter more than quality, a 512-token model is the efficient pick. A bigger context or dimension adds cost and latency with no gain.

EXAM TRAP

A larger-context embedding model is not better for chunks capped at 512 tokens; it only adds cost and latency.

Parse complex layouts first

Document Parsing handles tables, charts and mixed content. It reads PDFs, images, Word and PowerPoint files. Then use ai_prep_search downstream to get the parsed output ready for RAG.

Default limits of the ai_parse_document REST API

Limit	Value	Scope
Request rate	250 requests per minute	Per workspace
Pages	100 pages	Per document

Service limits that cap chunking

Rate and payload limits also shape your plan. Foundation and external model endpoints accept at most **4 MB** per request. GTE Large (En) defaults to **540,000** requests per hour, while BGE Large (En) and Qwen3-Embedding-0.6B allow **2,160,000**.

Keep retrieval consistent

Query with the embedding model that built the index. With a Databricks embedding endpoint, use the same model that created the index vectors. With direct-access indexes, or Delta Sync with self-managed embeddings, set a custom embedding model and text column on VectorSearchRetrieverTool.

Returning large chunks

Truncate big chunk columns in a retriever UDF. SQL clients may limit the rows or bytes they return. Trimming the column, for example with substring(chunked_text, 0, 8192), avoids row truncation.

Key takeaways

- Larger chunks and less overlap produce fewer chunks and fewer embeddings.
- Pick an embedding model whose context matches your chunk cap when cost matters most.
- ai_parse_document allows **100** pages per document and **250** requests per minute per workspace.

- With a Databricks embedding endpoint, query using the model that created the index vectors.

Sources: Databricks Exam Guide Databricks Certified Generative AI Engineer Associate (opens in a new tab) · Document Parsing | Databricks on AWS (opens in a new tab) · Foundation Model APIs limits and quotas | Databricks on AWS (opens in a new tab) · Connect agents to unstructured data | Databricks on AWS (opens in a new tab) · Use a custom embedding model with AI Search | Databricks on AWS (opens in a new tab)

Filter extraneous content in source documents that degrades quality of a RAG application

Start here

THINK FIRST

Which content should you strip from source documents before it reaches your RAG index?

Why it matters: Questions describe a noisy document set and ask which cleaning, filtering or deduplication step fixes it.

Noise in source documents lowers retrieval and answer quality. Parse documents well, clean out noise, filter unwanted documents and remove duplicates before chunks reach the index.

Key terms

Term	Meaning
Boilerplate	Repeated noise such as headers, footers and page numbers that cleaning removes.
Filtering	The pipeline step that removes irrelevant or unwanted documents from the collection.
Near duplicate	A document whose content nearly matches another; left in, it creates redundant chunks.
MinHash	A locality-sensitive hashing technique for finding near duplicates, available in Spark ML.

Start with good parsing

For PDFs and complex documents, use `ai_parse_document`. Poor parsing, such as missing tables or broken formatting, directly hurts retrieval quality. Review a sample of parsed output by hand on a regular basis. Store the raw source data in a target table for traceability and auditing.

Clean the extracted text

Remove boilerplate but keep the document's structure. Strip headers, footers, page numbers and special characters. Keep headings, lists and tables intact. Less malformed text means less noise for the chain to process.

EXAM TRAP

Cleaning does not mean removing all formatting. Headings, lists and tables are structure you should preserve.

Filter unwanted documents

Filtering drops documents your agent should not use. Documents may be irrelevant, too old, unreliable, harmful or sensitive. Every document source is also a possible path for data poisoning.

Problem documents and how to filter them

Problem	Example	Filter by
Harmful language	Toxic text in a document	A toxicity classifier's prediction
Sensitive information	Personal details you must not expose	A PII detection algorithm
Data poisoning	Documents planted by bad actors	Detection and filtering mechanisms
Irrelevant or outdated content	Documents off the agent's purpose	Metadata about each document

Remove duplicates

Duplicates create redundant chunks that hurt performance. Metadata alone catches some, such as the same title and creation date in different locations. For content-based near duplicates, use locality-sensitive hashing with MinHash in Spark ML.

EXAM TRAP

Keeping any one copy is not the best fix. Prefer the best version, such as the most recently updated or most authoritative one.

How MinHash finds matches

Hash the vectors with MinHash, then run a similarity join. The MinHash hashes measure Jaccard distance. A similarity join on those hashes returns each duplicate or near duplicate document.

Chunk to cut distraction

Smaller, focused chunks limit irrelevant content. Split large documents into semantically concentrated chunks. Retrieved data then fits the LLM's context, with less distracting information mixed in.

Key takeaways

- Use `ai_parse_document` for PDFs and complex documents; poor parsing hurts retrieval.
- Remove headers, footers and page numbers, but keep headings, lists and tables.
- Filter out irrelevant, outdated, harmful or sensitive documents before indexing.
- Find near duplicates with MinHash in Spark ML, and keep the best version.

Sources: AI Search retrieval quality guide | Databricks on AWS (opens in a new tab) · Build an unstructured data pipeline for RAG | Databricks on AWS (opens in a new tab)

Choose the appropriate Python package to extract document content from provided source data and format

Start here

THINK FIRST

How do you pick the right tool to turn a raw source file into text, fields or search chunks?

Why it matters: Questions describe source files and a goal, and ask which function or Python package extracts the content.

Parsing comes first: `ai_parse_document` turns raw files into a structured bronze layer. Other functions then work on that output, pulling fields, assigning labels or building search chunks. Scanned images need OCR, so `pytesseract` fits where HTML scrapers do not.

Key terms

Term	Meaning
Bronze layer	The standardized parsed output that captures text, tables or image descriptions, and document structure.
OCR	Needed to get text out of scanned images; `pytesseract` extracts image text with minimal code.
Semantic chunk	A piece of a parsed document enriched with titles, section headers and page references for AI Search.
ai_extract	Pulls structured fields from documents or plain text using a schema you define.

Parse raw files first

`ai_parse_document` **turns raw files into a bronze layer.** The layer captures text, tables or image descriptions, and document structure. The function reads its input from a Unity Catalog volume. It handles PDFs, images, Word and PowerPoint files, including complex layouts such as tables and charts.

EXAM TRAP

Parsing is not limited to PDFs. Word documents, PowerPoint files and images are supported too.

Pick the function for the goal

Each downstream function serves one goal. `ai_extract` and `ai_classify` work directly on parsed output, with no extra transformation step. Use `ai_prep_search` (Beta) downstream when the parsed output feeds RAG retrieval.

Three functions that work on parsed documents

Feature	ai_extract	ai_classify	ai_prep_search
What it does	Pulls structured fields using your schema	Assigns predefined categories	Builds semantic chunks for AI Search
Works on	Documents or plain text	Documents or text	Parsed documents
Detail to know	Runs directly on parsed output	Supports up to 500+ labels	Beta; adds titles, headers and page references

EXAM TRAP

`ai_classify` does not pull field values; it assigns predefined labels. Use `ai_extract` for structured fields.

Scanned images need OCR

For scanned .jpeg or .png files, use `pytesseract`. Text inside a scanned image needs OCR, and `pytesseract` extracts it with minimal code. HTML scraping tools such as `beautifulsoup`, `scrapy` and `pyquery` are not OCR tools.

EXAM TRAP

`beautifulsoup` sounds like a document parser, but it scrapes HTML. It cannot read text from a scanned image.

Find where a bad result came from

Missing or garbled text points to the parse or the source file. If the text parsed but a field is wrong, the problem is extraction instead.

EXAM TRAP

A wrong field value does not mean the parse failed. If the text is present in the parsed output, look at extraction.

Where and how parsing runs

The functions run on serverless compute or a SQL warehouse. The region must be a supported one. Parsing is also available in the Agent Bricks UI, the REST API and the Asynchronous API (Beta). A job task (Beta) writes results to a Unity Catalog table and processes only new or changed documents.

Limits to remember

REST API requests take up to 100 pages per document. The default REST API rate limit is **250** requests per minute per workspace. Information Extraction agents have a **128k** token maximum context length and do not support union schema types.

Key takeaways

- Parse first: `ai_parse_document` builds the bronze layer that later steps use.
- `ai_extract` pulls fields, `ai_classify` assigns labels and `ai_prep_search` builds search chunks.
- Scanned images need OCR: choose `pytesseract`, not an HTML scraper.
- Missing text means a parse or file problem; a wrong field means an extraction problem.

Sources: Intelligent document processing | Databricks on AWS (opens in a new tab) · Databricks Exam Guide Databricks Certified Generative AI Engineer Associate (opens in a new tab) · Document Parsing | Databricks on AWS (opens in a new tab) · Use Genie Code for intelligent document processing | Databricks on AWS (opens in a new tab) · Information Extraction | Databricks on AWS (opens in a new tab)

Define operations and sequence to write given chunked text into Delta Lake tables in Unity Catalog

Start here

THINK FIRST

Before your pipeline can write chunked text into a Delta table in Unity Catalog, what must be in place, and in what order?

Why it matters: Questions describe a chunk-writing step and ask which table type, namespace or privilege it needs.

Chunked text lands in a table addressed as catalog.schema.table. Writing it needs a chain of privileges, from the catalog down to the table.

Key terms

Term	Meaning
Metastore	The top-level container for Unity Catalog data, scoped to one cloud region.
Managed table	A table whose storage location Unity Catalog decides; the data still lives in your cloud account.
External table	A table stored at a path you specify; Unity Catalog manages only its metadata.
Usage privileges	USE CATALOG and USE SCHEMA, needed on the parents before any table privilege works.

Where the table lives

Address the target table as catalog.schema.table. A catalog holds schemas, and a schema holds tables, views, volumes and functions. One metastore covers a single cloud region and can serve several workspaces there. Volumes give file access and cannot be queried with SQL, so chunks you query belong in a table.

EXAM TRAP

A volume is not a place to query chunks with SQL; volumes give file-level access only.

Managed or external table

Prefer a managed table for new chunk tables. Databricks recommends managed tables so you get the latest table features.

Managed and external tables compared

Feature	Managed table	External table
Storage path	Decided by Unity Catalog	Specified by you
What Unity Catalog manages	Storage location; data stays in your cloud account	Metadata only, not lifecycle, optimization or layout

Privileges to write

Writes need a write privilege plus USE CATALOG and USE SCHEMA. Users start with no access to metastore data. Grant SELECT for reads and MODIFY for writes, or finer-grained INSERT, UPDATE and DELETE (Beta). Usage privileges on the parent catalog and schema are needed too. A grant on a schema applies to all its current and future tables.

EXAM TRAP

MODIFY on the table alone is not enough; without USE CATALOG and USE SCHEMA the write still fails.

Letting a team create tables

Grant CREATE TABLE on the schema plus both usage privileges. A team that creates the chunk table needs all three grants. The owner of an object holds every privilege on it and can grant them to others.

Example: Lets a group create tables in a schema and reach its parents.

```
GRANT CREATE TABLE ON SCHEMA main.default TO `finance-team`;
GRANT USE SCHEMA ON SCHEMA main.default TO `finance-team`;
GRANT USE CATALOG ON CATALOG main TO `finance-team`;
```

- `GRANT USE CATALOG ON CATALOG main`: usage on the parent catalog

Workspaces enabled by default

New workspaces come with Unity Catalog turned on. Workspaces created after **November 8, 2023** have Unity Catalog enabled automatically. There, all users get USE CATALOG on the workspace catalog and CREATE TABLE on its default schema. Separately, lineage records how data flows across tables, jobs and pipelines, down to columns.

Key takeaways

- Writing needs a write privilege such as MODIFY on the table, plus USE CATALOG and USE SCHEMA on its parents.
- Databricks recommends managed tables, whose storage path Unity Catalog decides.
- Schema-level grants cover all current and future tables in that schema.

Sources: Unity Catalog securable objects reference | Databricks on AWS (opens in a new tab) · Manage privileges in Unity Catalog | Databricks on AWS (opens in a new tab) · What is Unity Catalog? | Databricks on AWS (opens in a new tab) · Get started with Unity Catalog | Databricks on AWS (opens in a new tab)

Identify needed source documents that provide necessary knowledge and quality for a given RAG application

Start here

THINK FIRST

Given a RAG use case, how do you decide which documents belong in the corpus and which to leave out?

Why it matters: Questions describe a RAG application and its documents, and ask which ones to include, exclude or clean up.

The right corpus depends on what the application must answer. Include documents that cover those needs. Remove ones that are irrelevant, stale, duplicated or risky, and know the ingestion rules that skip files.

Key terms

Term	Meaning
Data corpus	The set of documents a RAG application can retrieve from to answer queries.
Near-duplicate	A document almost identical to another; it fills the index with redundant chunks.
MinHash	A locality-sensitive hashing technique in Spark ML that finds duplicate or near-duplicate documents.
Data poisoning	An attack that bad actors can launch through any document feed given to an agent.

Start from the use case

The right documents depend on the app's own requirements and goals. Without the right corpus, the app cannot retrieve what a query needs. For a customer support bot, consider knowledge base articles, FAQs, product manuals and specifications, and troubleshooting guides.

Bring in domain experts

Involve domain experts and stakeholders from the start. They help find and curate content that improves the corpus's quality and coverage. The corpus's composition is a key data pipeline consideration.

What to leave out

Exclude documents that are irrelevant, outdated, unreliable or harmful. Also hold back documents with sensitive information you do not want the agent to expose. Every document feed is a possible path for data poisoning.

EXAM TRAP

More documents are not always better. Context that is missing key information, or full of superfluous information, both make high-quality RAG output hard.

Remove duplicates

Near-duplicate documents create redundant chunks that hurt performance. MinHash, available in Spark ML, finds duplicates and near-duplicates by their content. Ingest the raw data into a target table first, for preservation, traceability and auditing.

Knowledge Assistant limits

Knowledge Assistant takes up to 10 knowledge sources. This legacy offering reads pdf, doc/docx, ppt/pptx, txt and md files from a Unity Catalog volume. Some files are skipped automatically and never reach the knowledge base.

Files Knowledge Assistant skips during ingestion

Item	Rule	Effect
Any file	Larger than 100 MB	Skipped
PDF, DOC, DOCX, PPT, PPTX	More than 500 pages; each slide counts as a page	Skipped
TXT and MD	No page limit	Never skipped for length
Any file	Name starts with _ or .	Skipped

EXAM TRAP

A **600**-slide PPTX is skipped even when the file is small, because each slide counts as a page.

Key takeaways

- Pick documents from the app's requirements; without the right corpus, retrieval fails.
- Leave out irrelevant, outdated, unreliable, harmful or sensitive documents.
- Deduplicate with MinHash to avoid redundant chunks in the index.

- Knowledge Assistant skips files over **100 MB** and documents over **500** pages.

Sources: Build an unstructured data pipeline for RAG | Databricks on AWS (opens in a new tab) · Use Knowledge Assistant to create a high-quality chatbot over your documents | Databricks on AWS (opens in a new tab) · Improve RAG application quality | Databricks on AWS (opens in a new tab)

Use tools and metrics to evaluate retrieval performance

Start here

THINK FIRST

Which metric and tool would you use to judge whether a retriever finds the right documents?

Why it matters: Questions describe a retrieval goal and ask which metric, tool or setup measures it.

Retrieval quality is measured apart from answer quality. AI Search can grade each retrieved document's relevance and compute ranking metrics from those grades. Each metric answers a different question about the results.

Key terms

Term	Meaning
Graded relevance	A score from **0** to **3** that an LLM judge gives each query and document pair.
DCG@10	A ranking metric that weighs both relevance and position, using the full 0 to 3 scale.
NDCG	DCG divided by the ideal DCG, giving a score from **0** to **1**.
RETRIEVER span	A trace span type that turns on MLflow's retrieval-specific LLM judges.

Measure retrieval and response

Collect retrieval metrics and response metrics together. A RAG app can answer badly even with the right context retrieved. It can also answer well on top of faulty retrieval.

Built-in evaluation in AI Search

AI Search compares search strategies on your own data. This Beta feature needs a managed Delta Sync index. Every generated query runs with ANN, hybrid and full-text retrieval, and each strategy is tried with and without the reranker.

EXAM TRAP

You do not need to own the index. Anyone with query access can start a run, and that user, not the index owner, owns the job.

How results are graded

An LLM judge scores each query and document pair on a 4-point scale. Scores run from **0** to **3**, and a score of **2** or higher counts as relevant. Every metric carries **95%** confidence intervals, so you can tell whether strategy differences are meaningful.

Pick the metric for the goal

Use DCG@10 as the primary metric of overall retrieval quality. The other metrics answer narrower questions, so choose the one that matches your goal.

Retrieval metrics and what each tells you

Metric	What it tells you	Detail
DCG@10	Relevance and ranking position together	The recommended primary metric
NDCG	DCG divided by the ideal DCG	Ranges from 0 to 1; 1.0 is perfect order
MRR	Average of 1/rank of the first relevant result	First relevant means a score of 2 or more
Precision@k	Share of the top-k results that are relevant	Relevant means a score of 2 or more
Recall@k	Completeness, for RAG where a missed document means an incomplete answer	Needs ground truth listing every relevant item

EXAM TRAP

Recall cannot be computed from a partial answer key. The ground truth has to contain every relevant item.

How DCG scores a result

Each result adds a gain of `2^relevance - 1`**.** That gain is discounted logarithmically by position, so the same relevant result is worth less when it ranks lower.

Tracing for MLflow judges

Mark the retrieval function with `span_type="RETRIEVER"`**.** This turns on MLflow's retrieval-specific LLM judges for that part of your RAG app.

Key takeaways

- DCG@10 is the recommended primary metric for overall retrieval quality.
- Use Recall@k when completeness matters; it needs ground truth with every relevant item.
- A score of **2** or more counts as relevant for Precision@k and MRR.
- Collect retrieval and response metrics together.

Sources: Assess performance: Metrics that matter | Databricks on AWS (opens in a new tab) · Evaluate AI Search retrieval quality | Databricks on AWS (opens in a new tab) · Tutorial: Evaluate and improve an agent |

Databricks on AWS (opens in a new tab)

Design retrieval systems using advanced chunking strategies

Start here

THINK FIRST

How do you choose a chunking strategy that keeps answers precise without losing the context around them?

Why it matters: Questions describe documents and a retrieval goal, and ask which chunk size, strategy or enrichment fits.

Chunk size trades precision for context. Advanced approaches such as semantic chunking take more effort but can have a bigger impact. Rich metadata and high-quality parsing matter more than over-optimizing chunk size.

Key terms

Term	Meaning
Semantic chunking	Grouping sentences by similarity, with embeddings finding natural boundaries, instead of by fixed size.
Parent-child chunking	Searching small child chunks for precision, then returning their larger parent chunks for context.
Semantic metadata	Context added to chunks, such as document summaries and section headers.
Boilerplate	Repeated page clutter such as headers, footers and page numbers, removed before chunking.

Size is a trade-off

Small chunks pinpoint facts; large chunks keep context. A **512**-token chunk is the balanced starting point. Whatever the size, the retrieved chunks together must fit the LLM's context window.

Smaller and larger chunks compared

Feature	Smaller chunks	Larger chunks
Strength	Better localization of specific facts	More context preserved
Weakness	May lose context	Harder to pinpoint relevant information
Starting point	256 tokens, for precise fact retrieval	1024 tokens, for more context per chunk

EXAM TRAP

There is no proven best chunk size. It is still an active research area, and embeddings can miss basic information in long contexts.

Semantic chunking

Semantic chunking groups sentences by meaning, not fixed size. Embeddings find natural boundaries, so related ideas stay together. Like the other advanced approaches, it costs more work, yet its payoff can be larger.

Parent-child chunking

Search small chunks, then return their large parents. Each child chunk, about **512** tokens, is stored with its parent of about **2048** tokens. The child gives a precise match, and the parent gives the LLM context.

Example: Splits each document into large parent chunks, then small child chunks stored with their parent's text.

```
for parent_chunk in create_chunks(doc, size=2048):
  for child_chunk in create_chunks(parent_chunk, size=512):
    source_table.append({
      "text": child_chunk,
      "parent_text": parent_chunk,
    })
```

- `size=2048`: a large parent chunk, kept for context
- `size=512`: a small child chunk, used for precise search

Parse and clean first

Use `ai_parse_document` for PDFs and complex documents. Poor parsing, such as missing tables or broken formatting, directly hurts retrieval. Strip boilerplate such as headers, footers and page numbers. Keep headings, lists and tables intact.

Enrich chunks with metadata

Metadata often beats over-tuning chunk size. Extract entities, topics and categories for precise filtering. Prepend the document title, summary and section name to each chunk. Semantic metadata in separate columns needs `columns_to_rerank`, while keyword-only metadata needs hybrid search in full-text mode.

EXAM TRAP

Metadata in its own column is not used automatically. Semantic fields need `columns_to_rerank`, and keyword fields need hybrid search.

Evaluate the parts

Test retrieval on its own, not only the whole app. Formatting changes can shift which chunks come back. Each retriever result carries `doc_uri` as its primary document identifier.

Key takeaways

- Start chunk-size tests at **256**, **512** and **1024** tokens.
- Parent-child chunking searches small chunks and returns their parents.

- Clean parsing and rich metadata matter more than over-tuning chunk size.
- The retrieved chunks together must fit the LLM context window.

Sources: AI Search retrieval quality guide | Databricks on AWS (opens in a new tab) · RAG (Retrieval Augmented Generation) on Databricks | Databricks on AWS (opens in a new tab) · Connect agents to unstructured data | Databricks on AWS (opens in a new tab)

Explain the role of re-ranking in the information retrieval process

Start here

THINK FIRST

Where does re-ranking fit between retrieval and generation, and when is its extra latency worth paying?

Why it matters: Questions describe a retrieval setup with its latency or traffic needs, and ask whether and how to rerank.

Retrieval queries a knowledge base for supporting data for the LLM. Adding the reranker to that search boosts precision, but it adds time. So the choice depends on traffic and the latency budget.

Key terms

Term	Meaning
Reranker	An option on an AI Search similarity search that reranks the results to boost precision.
Cross-encoder	A reranking model type that can give a meaningful quality lift; a lightweight example is a TinyBERT model.
DCG@10	A retrieval metric that reflects how relevant the top results are and where they rank.
NDCG	DCG divided by the ideal DCG: how close the order is to the best possible order.
QPS	Queries per second. The built-in reranker handles about **5** out of the box.

Where reranking sits

Reranking is switched on inside the retrieval search call. In a RAG agent, retrieval uses the request to fetch supporting data from a vector store, a keyword search or a SQL database. Reranking then raises precision, reaching high recall with fewer candidates. Pair it with hybrid search and filtering for the best results.

EXAM TRAP

Reranking alone is not the whole recipe. It works best when combined with hybrid search and filtering.

The built-in reranker

Databricks offers a built-in reranker as a one-line change. It gives roughly **15%** better quality. In typical workloads it reranks **50** results in under **1** second, and in about **250** msec for shorter chunks.

Example: Turns on the built-in reranker in a similarity search and names the columns it scores.

```
reranker={
    "model": "databricks_reranker",
    "parameters": {
        "columns_to_rerank": ["text", "parent_doc_summary"]
    }
}
```

- `"databricks_reranker"`: the built-in reranking model
- `columns_to_rerank`: the columns the reranker reads when it scores results

When the built-in reranker fits

It suits RAG agents and quality-first apps at modest traffic. In a RAG agent, LLM generation dominates the latency, which is why RAG agents are listed as a good fit.

Good and poor fits for the built-in reranker

Feature	Good fit	Poor fit
Application type	RAG agents and quality-first applications	Real-time search bars needing under 100 msec
Traffic	Low to moderate, about 5 QPS out of the box	Above 5 QPS without additional scaling

EXAM TRAP

The built-in reranker is the wrong choice for a search bar that needs under **100** msec, or for traffic above **5** QPS without extra scaling.

Faster reranking for search bars

Serve a lightweight reranker on Model Serving for sub-100 msec speed. Search bars and high-QPS apps can still gain quality from reranking. Deploy a small cross-encoder such as `ms-marco-TinyBERT-L-2-v2`, served as a custom model.

Give the reranker more context

Semantic metadata, such as document summaries, helps the reranker score. It also adds semantic signal for embedding models. When that metadata sits in separate columns, list them in `columns_to_rerank` so reranking takes them into account.

EXAM TRAP

Adding a summary column is not enough by itself. Reranking uses it only when the column is listed in `columns_to_rerank`.

Measuring the gain

Evaluation tests each query with and without the reranker. Each generated query runs with ANN, hybrid and full-text retrieval. DCG@10 is the recommended primary metric, since it reflects both relevance and rank position. If the reranker lifts metrics significantly, turn it on when the latency budget allows.

Key takeaways

- Reranking boosts precision by reaching high recall with fewer candidates.
- The built-in reranker gives about **15%** better quality for a one-line change.
- Use it for RAG agents near **5** QPS; use a lightweight custom reranker for sub-**100** msec needs.
- List metadata columns in `columns_to_rerank` so the reranker reads them.

Sources: RAG (Retrieval Augmented Generation) on Databricks | Databricks on AWS (opens in a new tab) · AI Search retrieval quality guide | Databricks on AWS (opens in a new tab) · Evaluate AI Search retrieval quality | Databricks on AWS (opens in a new tab)

Domain 3 | 30% of the exam | 37 min read

Application Development

Overview

This domain is about 30% of the exam, roughly 14 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Select Langchain/similar tools for use in a Generative AI application

Start here

THINK FIRST

Faced with an agent design, how do you choose between LangChain, LangGraph, OpenAI SDKs, MCP servers and Databricks tools?

Why it matters: Questions describe an agent's task and stack, and ask which library, tool or tracing call fits it.

Pick a framework for the agent, a tool for each data need, and the tracing call that makes it visible. Databricks pairs LangChain and OpenAI libraries with AI Bridge tools, MCP servers and MLflow autologging.

Key terms

Term	Meaning
Autologging	Automatic MLflow tracing for libraries such as LangChain, LlamaIndex and OpenAI.
AI Bridge	The recommended way for new Python agents to query Databricks AI Search indexes.
MCP server	A tool server an agent calls. Databricks-provided and registered ones use Unity Catalog permissions.
Retriever schema	The standard shape for retriever output: a `List[Document]` with extra attributes in `metadata`.

Agent frameworks

Build custom agents with the OpenAI client, the OpenAI Agents SDK or LangGraph. Any of these can call MCP tools. Databricks-provided and registered MCP servers are governed by Unity Catalog permissions.

Picking an MCP server

Match the MCP server to the kind of question. Plain-language business questions and hand-written SQL call for different servers.

Which Databricks MCP server to use

Need	Server	Detail
Business questions in natural language	Genie One MCP server	Grounds answers in Genie Ontology, the governed semantic layer
Specific SQL queries	Databricks SQL MCP server	Also suits development and data engineering

EXAM TRAP

For a business user's plain-English question, the Databricks SQL MCP server is not the recommended pick. Genie One is.

Retrieval tools

New agents query AI Search indexes through AI Bridge. `VectorSearchRetrieverTool`, in `databricks-langchain` or `databricks-openai`, turns an index into an agent tool. Direct-access indexes, and Delta Sync indexes with self-managed embeddings, also need a custom embedding model and text column.

EXAM TRAP

With self-managed embeddings, `VectorSearchRetrieverTool` is not ready with defaults alone. You must set a custom embedding model and a text column.

Retriever output

Return a `List[Document]`, with extras in the `metadata` field. Attributes such as `doc_uri` and `similarity_score` go in `metadata`. For new retrievers, `mlflow.models.set_retriever_schema` is deprecated. Wrapping a Unity Catalog retriever function with `UCFunctionToolkit` creates RETRIEVER spans for tracing.

Tracing LangChain

`mlflow.langchain.autolog()` **turns on LangChain tracing.** On serverless compute you must call it yourself, because autologging is off there by default. Traced APIs include `invoke`, `batch`, `stream`, their async forms and `get_relevant_documents`.

Works: Turns on automatic LangChain tracing, which serverless compute needs you to call explicitly.

```
mlflow.langchain.autolog()
```

Fails: Passing disable=True switches LangChain auto tracing off globally instead of on.

```
mlflow.langchain.autolog(disable=True)
```

Production and custom traces

For production, choose the `mlflow-tracing` package, which is optimized for it. To change what a LangChain trace holds, write a callback handler that inherits from `MlflowLangchainTracer`. Manual `@mlflow.trace` decorators work alongside autologging.

Key takeaways

- New agents on AI Search use AI Bridge's `VectorSearchRetrieverTool`.
- Use Genie One MCP for business questions and Databricks SQL MCP for SQL work.
- On serverless compute, call `mlflow.langchain.autolog()` explicitly.
- Retrievers return `List[Document]`; `set_retriever_schema` is deprecated.

Sources: Connect agents to unstructured data | Databricks on AWS (opens in a new tab) · Call MCP tools from a Python agent | Databricks on AWS (opens in a new tab) · MCPs | Databricks on AWS (opens in a new tab) · Tracing LangChain | Databricks on AWS (opens in a new tab)

Qualitatively assess responses to identify common issues such as quality and safety

Start here

THINK FIRST

When a RAG answer looks wrong, how do you tell whether retrieval, generation or safety is the real problem?

Why it matters: Questions show a poor or unsafe response and ask which issue it reveals or which judge would catch it.

A response can fail on accuracy, grounding or safety, and the cause can sit in retrieval or in generation. Collect both retrieval and response metrics, then use traces to find the step that went wrong.

Key terms

Term	Meaning
Groundedness	Whether a response is supported by its retrieved context or is a hallucination.
Hallucination	Output not grounded in the retrieved context; one way generation quality fails.
Ground truth	Human-labeled correct answers that some judges, such as correctness, compare against.
Adversarial example	A test question that attempts prompt injection or tries to draw out an inappropriate response.

Term	Meaning
Evaluation set	The questions used to judge quality: at least **30**, ideally **100** to **200**.

Three response checks

Judge responses on correctness, groundedness and safety. Response metrics cover accuracy against ground truth and grounding in the retrieved context. They also cover safety, meaning no toxicity.

Three response judges compared

Judge	What it asks	Needs ground truth
correctness	Did the agent give a correct response?	Yes
groundedness	Is the response grounded in context, or made up?	No
safety	Does the response contain harmful content?	No

EXAM TRAP

Groundedness and safety judges need no ground truth. Of these three, only correctness compares outputs with human labels.

Spotting generation problems

Generation issues show as hallucinations, inconsistency or off-target answers. Off-target means the reply does not directly address the user's query. The context matters too: missing or superfluous information makes good output hard to produce.

Retrieval or response?

Collect both retrieval and response metrics. A RAG app can answer poorly even when it retrieved the right context. It can also answer well from faulty retrieval. Retrieval quality depends on pipeline levers, such as chunking, metadata and the embedding model. It also depends on chain levers, such as query transformation, chunk count and re-ranking.

EXAM TRAP

A good final answer does not prove retrieval worked. Faulty retrieval can still yield good responses, so check retrieval metrics too.

Trace to the failing step

Traces show which intermediate step caused a failure. The input and final response alone cannot reveal that. With traces you inspect every step, then turn typical failures into evaluation datasets.

Build a tough evaluation set

Use at least 30 questions, ideally 100 to 200. Include adversarial examples, such as prompt injection attempts or requests for inappropriate responses. Stakeholders can give feedback in a web chat interface. Their feedback and the outputs are saved in Delta tables for curation.

Judges without labels

Agent Evaluation can score quality with no ground truth. When ground truth exists, it adds metrics such as answer correctness. Any LLM judge must be tuned to the use case: find where it fails, then improve it there.

EXAM TRAP

An LLM judge is not reliable out of the box for every use case. It must be tuned on the cases where it does not work well.

Key takeaways

- Correctness needs ground truth; groundedness and safety do not.
- Hallucinations, inconsistent output and off-target answers point to generation issues.
- Measure retrieval and response separately: one can be good while the other fails.
- Start with at least **30** evaluation questions, including adversarial ones.

Sources: Assess performance: Metrics that matter | Databricks on AWS (opens in a new tab) · What is agent observability and quality? | Databricks on AWS (opens in a new tab) · Improve RAG application quality | Databricks on AWS (opens in a new tab) · Define “quality”: Evaluation sets | Databricks on AWS (opens in a new tab)

Select chunking strategy based on model & retrieval evaluation

Start here

THINK FIRST

How would you decide whether smaller or larger chunks will serve your RAG application better?

Why it matters: Questions describe a retrieval problem and ask which chunking strategy or evaluation metric fits it.

Chunk size trades precision against context, and no size is right in advance. Build a reproducible evaluation first, then compare strategies by how retrieval metrics change.

Key terms

Term	Meaning
Semantic chunking	Splitting text into groups of similar sentences instead of fixed-size pieces.

Term	Meaning
Parent-child chunking	Small-to-big retrieval: small child chunks for precision inside large parent chunks for context.
Recall@k	Of all known relevant documents, the share retrieved within the top k, scored from 0 to 1.
DCG@10	A ranking metric that scores result relevance together with each result's rank position.

Small versus large chunks

Small chunks pinpoint facts; large chunks keep context. Neither wins everywhere, so test sizes against your own questions. Whatever size you pick, the retrieved chunks must still fit in the LLM context window together.

Chunk sizes commonly tested and what each favors

Feature	Small	Medium	Large
Typical size	**256**	**512**	**1024**
Strength	Precise fact retrieval	A balance of both	More context per chunk

EXAM TRAP

Bigger is not always better. Large chunks keep context but make the relevant passage harder to find, and many must fit the context window.

Strategies beyond fixed size

Semantic and parent-child chunking avoid one fixed size. Semantic chunking groups sentences that are similar. Parent-child chunking stores small child chunks, for example **512**, inside large parents, for example **2048**, so a precise match returns its wider context.

Clean the data first

Good chunks start from clean, structured text. Remove boilerplate such as headers, footers and page numbers. Keep headings, lists and tables intact, and cut along semantic boundaries.

Often better than resizing

Metadata and parsing often beat chunk-size tuning. Extract entities, topics and categories for precise filtering. Parse documents cleanly, for example with `ai_parse_document`, and add summaries and section headers to chunks.

EXAM TRAP

Endless chunk-size tweaking is rarely the best lever. Metadata extraction, quality parsing and semantic metadata usually help more.

Measure before you optimize

Set up a reproducible evaluation system first. The data need not be perfect. What matters is an automated way to measure change, and judging strategies by relative improvement rather than absolute scores.

Choosing the metric

Track Recall@k when missing context is the risk. In a RAG agent, a missed document leads to wrong answers or hallucinations. For overall retrieval quality, DCG@10 is the recommended primary metric because it scores relevance and ranking position.

Two retrieval metrics compared

Feature	Recall@k	DCG@10
Measures	Share of relevant documents in the top k	Relevance and position in the ranking
Use it for	Completeness, as in RAG	Primary measure of overall retrieval quality

Judging relevance and components

Evaluate retrieval on its own, not only end to end. A formatting change can alter which chunks come back. AI Search retrieval quality evaluation (Beta) has an LLM judge rate each query and document pair on a **4**-point relevance scale.

Key takeaways

- Smaller chunks localize facts; larger chunks keep context but blur the target.
- Common sizes to test are **256**, **512** and **1024**.
- For RAG, track Recall@k; for overall retrieval quality, DCG@10 is the primary metric.
- Have a reproducible evaluation in place before you tune retrieval.

Sources: AI Search retrieval quality guide | Databricks on AWS (opens in a new tab) · Evaluate AI Search retrieval quality | Databricks on AWS (opens in a new tab) · Introduction to evaluation & monitoring RAG applications | Databricks on AWS (opens in a new tab)

Augment a prompt with additional context from a user's input based on key fields, terms, and intents

Start here

THINK FIRST

How does a user's question become a prompt that carries the right supporting facts for the LLM?

Why it matters: Questions describe a user request and ask how to retrieve context and insert it into a prompt template.

A RAG chain understands the question, retrieves supporting data, augments the prompt with it and then generates the answer. A prompt template with variables is where the user's input and the retrieved context meet.

Key terms

Term	Meaning
Augmentation	Combining retrieved data with the user's request, often in a template, to build the prompt.
Template variable	A placeholder in double braces, such as `{{name}}`, filled with values at run time.
Chat prompt	A list of role-based messages, such as system and user, for conversational models.
Knowledge base	The outside store queried during retrieval: a vector store, keyword search or SQL database.

The RAG chain in order

Retrieve, then augment, then generate. The chain first works out what the user's question means. It then retrieves supporting data, augments the prompt with that data and has an LLM generate a response from the augmented prompt.

What each step of the RAG chain does

Feature	Retrieval	Augmentation	Generation
Input	The user's request	Supporting data plus the request	The augmented prompt
Output	Supporting data from a knowledge base	A prompt, often built from a template	The LLM's response

Retrieving the right context

The user's request drives the query. Use the request to search a vector store, keyword search or SQL database. Retrieval can also filter personal or proprietary data by the user's credentials.

What context can add

RAG brings in knowledge the LLM never saw. Memos, emails and other documents outside the training data can answer domain-specific questions. The supporting data can be structured or unstructured, depending on the use case.

Templates with variables

Double braces mark each value filled in at run time. Both simple and conversation prompts use `{{name}}` style variables. A Chat prompt splits instructions and input into role-based messages, such as system and user.

EXAM TRAP

Single braces are not the registry's variable syntax. MLflow can convert prompts to single braces for LangChain and LlamaIndex, which use f-string style.

Filling the template in code

Load the prompt, then call `format()` with the user's values. `mlflow.genai.load_prompt()` fetches a registered prompt by its URI. Passing `allow_missing=True` when loading lets optional parameters stay empty.

Example: Loads a registered prompt and fills its question variable with user input.

```
my_uri = "prompts:/docs.default.customer_support@production"
prompt = mlflow.genai.load_prompt(name_or_uri=my_uri)
formatted_prompt = prompt.format(
    question="How do I reset my password?"
)
```

- `prompt.format(`: fills each template variable by name

EXAM TRAP

Loading the prompt does not insert the user's input. You still call `format()` with a value for each variable.

Key takeaways

- Augmentation joins retrieved data and the user's request into one prompt.
- Registry templates mark variables with double braces, like `{{name}}`.
- Fill a loaded prompt by calling `format()` with the variable values.
- Retrieval can limit personal or proprietary data based on user credentials.

Sources: RAG (Retrieval Augmented Generation) on Databricks | Databricks on AWS (opens in a new tab) · Prompt Registry | Databricks on AWS (opens in a new tab) · Create and edit prompts | Databricks on AWS (opens in a new tab) · Prompt Registry examples | Databricks on AWS (opens in a new tab)

Create a prompt that adjusts an LLM's response from a baseline to a desired output

Start here

THINK FIRST

How do you move an LLM's answers from a rough baseline to the output you need?

Why it matters: Questions show a baseline response and ask which prompt change, format setting or tool reaches the target.

Start with a simple prompt and improve it based on evaluation results. Version each change in the Prompt Registry, and use structured outputs when the answer must follow a fixed format.

Key terms

Term	Meaning
Prompt Registry	MLflow's store for prompt templates, with immutable versions that keep history and allow rollback.
Structured outputs	A way to make a chat model return text, unstructured JSON or JSON that follows a schema.
response_format	The request field that sets structured output the same way for any supported chat model.
Prompt optimization	Tuning a prompt with data that holds expected responses and facts, for example with GEPA.

Start simple, then iterate

Begin with a basic prompt and improve it from evaluation results. Prompt optimization can help steer behavior toward your use case. It uses data with expected responses and facts. MLflow can optimize a simple prompt with the GEPA optimizer.

Versioning prompts

Prompt versions are immutable; an edit creates a new version. This keeps full history and lets you roll back. Templates fill variables at runtime with `{{variable_name}}` syntax.

Text and Chat prompt types compared

Feature	Text	Chat
Template shape	A single text template	A list of role-based messages, such as system and user
Use it for	Completion-style prompts	Chat-style prompts for conversational models

EXAM TRAP

You cannot edit a saved prompt version in place. Any change becomes a new version.

Comparing versions

The Compare button shows only a text diff. To learn which version gives better outputs, evaluate both with the same dataset and judges. Then compare the evaluation runs.

EXAM TRAP

A text diff does not tell you which prompt performs better. To find out, run evaluations on both versions with the same dataset and judges, then compare the runs.

Forcing a format

Structured outputs use `response_format` on any supported chat model. Databricks handles the translation for each provider. They work on pay-per-token and provisioned throughput endpoints. The techniques behind them raise token use, which affects billing.

Schema limits

A JSON schema can hold at most 64 keys. Flatten heavily nested schemas for better quality. The `pattern` keyword is not supported, and neither is composition with `anyOf`, `oneOf`, `allOf`, `prefixItems` or `$ref`.

EXAM TRAP

Deep nesting does not improve structured output. Heavily nested schemas lower generation quality, so flatten them when you can.

Claude model specifics

Claude models accept `json_schema`, not `json_object`. Omit `response_format` for unconstrained output. Structured outputs do not work with streaming, so set `stream` to false.

EXAM TRAP

Streaming and structured outputs do not mix for Claude models. Set `stream` to false whenever you pass `response_format`.

Key takeaways

- Start with a basic prompt and refine it based on evaluation results.
- Prompt versions are immutable, so every edit creates a new version you can roll back from.
- Compare prompt versions with evaluation runs on the same dataset and judges, not the text diff.
- Keep JSON schemas flat and at **64** keys or fewer.

Sources: Evaluate and compare prompt versions | Databricks on AWS (opens in a new tab) · Create and edit prompts | Databricks on AWS (opens in a new tab) · Structured outputs on Databricks | Databricks on AWS (opens in a new tab) · Optimize prompts tutorial | Databricks on AWS (opens in a new tab)

Implement LLM guardrails to prevent negative outcomes

Start here

THINK FIRST

How would you stop an LLM app from leaking secrets or following a jailbreak, both in evaluation and at the gateway?

Why it matters: Questions describe a risk such as PII, secrets or jailbreaks and ask which guardrail scorer or gateway policy blocks it.

Guardrails work at two points. Guardrails AI scorers in MLflow check outputs and inputs with rules, without LLM calls. Unity Gateway service policies sit in the request path and can block interactions as they happen.

Key terms

Term	Meaning
Guardrails AI scorer	A Guardrails validator used as an MLflow scorer for rule-based checks.
Service policy	A Unity Gateway policy, in Beta, that can inspect and block gateway traffic.
Fail closed	Deny the interaction whenever a check errors or returns something unexpected.
Rank	Sets policy order: the lowest rank runs first on requests and last on responses.

Rule-based scorers

Guardrails validators become MLflow scorers with no LLM calls. Install the guardrails-ai package first. Pass validator settings, such as a threshold, as keyword arguments to the constructor, or build a scorer by name with get_scorer.

Which validator for which risk

Pick the validator that matches the harm you fear. Note which side each one checks: the jailbreak check reads the **input**, while the others read the **output**.

Guardrails AI validators and what they detect

Validator	Checks	Detects
SecretsPresent	Output	API keys, tokens or other secrets
DetectPII	Output	Personally identifiable information
DetectJailbreak	Input	Jailbreak or prompt injection attempts

EXAM TRAP

DetectJailbreak does not scan the model's output; it looks for jailbreak or prompt injection attempts in the input.

Gateway service policies

Workspace admins turn on service policies as a preview. Opt into the Unity Gateway Service Policies preview. While you review a policy, Log mode records its decisions but blocks nothing. With several policies, Rank sets the order.

Bedrock Guardrails as a policy

A Python policy calls Bedrock ApplyGuardrail on each message. It blocks the interaction when Bedrock intervenes, and it fails closed on errors. A function that declares a service credential must set environment_version = '6', or it fails with CREDENTIALS_UNSUPPORTED_ENVIRONMENT.

EXAM TRAP

A Python policy cannot redact; a Bedrock guardrail that would mask sensitive data denies the whole interaction instead.

Limits on MCP services

On MCP, the policy checks tool calls, not tool results. Before a tool runs, the policy passes its name and arguments to Bedrock as input. Anything the tool returns goes unchecked by this policy.

EXAM TRAP

The Bedrock policy does not inspect MCP tool results, only the tool name and arguments sent before the call.

Key takeaways

- Guardrails AI scorers give rule-based checks in MLflow without LLM calls.
- DetectJailbreak checks inputs; SecretsPresent and DetectPII check outputs.
- Set a policy's Mode to Log to record decisions without blocking.
- Python policies cannot redact, so masking guardrails deny the interaction.

Sources: Guardrails AI scorers | Databricks on AWS (opens in a new tab) · Tutorial: Enforce Amazon Bedrock Guardrails with a custom Python policy | Databricks on AWS (opens in a new tab)

Select the best LLM based on the attributes of the application to be developed

Start here

THINK FIRST

When should you serve your own LLM instead of using a model Databricks already hosts?

Why it matters: Questions describe an application's model needs and ask which serving option or GPU type fits.

Use Foundation Model APIs when they serve the model you need, unchanged. Serve a custom LLM only for models they do not support, such as your own fine-tuned LLM, and size the GPU to the model.

Key terms

Term	Meaning
Foundation Model APIs (FMAPI)	The option to use when it serves the model you need unchanged; it is simpler to use and highly optimized.
Custom LLM serving	Serving your own open model on a GPU endpoint, currently in Public Preview.
LoRA adapter	An adapter applied to a base model; one endpoint can serve several on the same base model.
Scale-to-zero	Stopping all replicas when an endpoint is idle, so the next request waits for startup.

FMAPI first

If FMAPI serves your model unchanged, use FMAPI. It is simpler to use and highly optimized. System-provided model services are open to all account users by default. You can also chat with them in AI Playground.

> **EXAM TRAP**
> Custom serving is not the default for a standard model. When FMAPI already serves that model unchanged, FMAPI is the advised path.

When to serve your own

Custom serving covers models FMAPI does not support. Examples are a new LLM, a speech-to-text model, an embedding model, or an LLM you fine-tuned on AI Runtime. This feature is in Public Preview.

> **EXAM TRAP**
> Custom serving does not fit every model. Skip it when the model does not fit on a single GPU with about **80 GB** of memory.

Matching GPU to model size

Pick the workload type by the model's size. GPU_XLARGE is recommended for large LLMs and has region limits. It also does not support scale-to-zero.

GPU workload types for custom LLM serving

Feature	GPU_MEDIUM	GPU_XLARGE
Hardware	**1x** A10G, **24 GB**	**1x** H100, **80 GB**
Best for	Models up to about **10B** parameters	Large LLMs

Feature	GPU_MEDIUM	GPU_XLARGE
Availability	Generally available	**us-west-2** only, with enrollment

Sizing for traffic

Size `workload_size` for your peak traffic. Custom LLM endpoints currently do not autoscale dynamically. For production traffic, turn off scale-to-zero. An idle endpoint takes one to several minutes to start again.

EXAM TRAP

Scale-to-zero is a poor fit for production. The first request after an idle period waits while the model loads again.

Serving many fine-tunes

One endpoint can serve several LoRA adapters on one base model. vLLM loads the base model one time. It then applies whichever adapter each request names.

Custom LLM agents are legacy

The Custom LLM agent offering is legacy. For custom text tasks on your data, use AI Functions such as `ai_query`. Optimizing a Custom LLM agent needs at least **100** inputs. It weighs cost against quality with MLflow and Agent Evaluation.

Key takeaways

- Use FMAPI when it serves your model unchanged; it is simpler and highly optimized.
- Serve a custom LLM for unsupported open models or an LLM you fine-tuned yourself.
- GPU_MEDIUM suits models up to about **10B** parameters; GPU_XLARGE suits large LLMs.
- Custom LLM endpoints do not autoscale, so size for peak and turn off scale-to-zero in production.

Sources: Serve custom LLMs with Custom Model Serving | Databricks on AWS (opens in a new tab) · Get started querying LLMs on Databricks | Databricks on AWS (opens in a new tab) · Use Custom LLM to create an AI agent for text (legacy) | Databricks on AWS (opens in a new tab)

Select an embedding model context length based on source documents, expected queries, and optimization strategy

Start here

THINK FIRST

How should chunk size, expected queries and cost goals steer your choice of an embedding model's context length?

Why it matters: Questions give a chunk size and a priority such as cost or quality, and ask which embedding

model fits best.

An embedding model's context length should fit the chunks you feed it. A bigger window is not free, because larger context or dimension models add cost and latency. Chunk the documents to fit, then, when cost leads, pick the model that just covers the chunk size.

Key terms

Term	Meaning
Context length	The most tokens an embedding model accepts in one input. GTE accepts up to **8192**.
Chunking	Splitting documents into smaller pieces so each one stays within the model's context limit.
Embedding dimension	A size property of an embedding model, such as **384** dimensions. Larger dimension models add cost and latency.
Cold start	The startup overhead of a Model Serving endpoint, felt most when it has scaled down to zero.

Fit chunks to the window

Chunk documents so no piece goes past the model's context limit. The GTE model accepts up to **8192** tokens. Even so, smaller chunks are recommended. They let a RAG application pass a wider variety of examples to the reasoning model.

EXAM TRAP

A large window does not mean each chunk should fill it. Smaller chunks give the reasoning model more varied examples, even with GTE's **8192**-token limit.

Match the model to the chunks

Capped chunks call for a model whose window just covers them. Say chunks stop at **512** tokens, and cost and latency outweigh quality. Then a **512**-context model is the efficient pick, for example one of **0.13GB** with **384** dimensions. A larger context or dimension adds cost and latency with no benefit.

EXAM TRAP

Picking the largest window to be safe is the classic wrong answer. With **512**-token chunks and cost first, a bigger model only adds expense and delay.

Ways to get a model

Use a hosted model, serve an open one, or log your own. A served open model such as e5-small-v2 can be used with AI Search, and so can your own MLflow-logged model. Foundation Model APIs offer gte-large-en pay-per-token.

Three ways to get an embedding model

Option	What you do	Example
Foundation Model APIs	Call the pay-per-token endpoint	gte-large-en
Open source model	Serve it on a Model Serving endpoint	e5-small-v2
Your own model	Log it with MLflow as a custom Python model	Any model with the expected signature

Custom model signature

A custom embedding model takes one string and returns a tensor. Its input schema is a single ColSpec string. Its output schema is TensorSpec, and the embeddings come back in a NumPy array.

Wiring the model into an index

A managed-embeddings index names a text column and an endpoint. In a Delta Sync index, set `embedding_source_column` to the text column. Set `embedding_model_endpoint_name` to the serving endpoint. In the GTE example, the chunked Delta table is written with `delta.enableChangeDataFeed` set to true.

Serving and querying

Frequent queries favor GPU endpoints that stay above zero. Each Model Serving endpoint pays a cold start overhead, so avoid scaling down to **0** when queries are frequent. In LangChain, import `DatabricksEmbeddings` and set its `endpoint` parameter. From outside the workspace, use the OpenAI client directly. The `ai_query` SQL function is in Public Preview.

> **EXAM TRAP**
> Letting an embedding endpoint scale to zero is a poor fit for frequent queries, because Model Serving endpoints carry cold start overhead.

Key takeaways

- Chunk documents so each piece fits the embedding model's context limit.
- With **512**-token chunks and cost first, a small **512**-context model beats larger ones.
- Smaller chunks give the reasoning model a wider variety of examples, even when the window is larger.
- Custom models are logged with MLflow: one ColSpec string in, TensorSpec out.

Sources: AI Search foundational embedding model (GTE) example | Databricks on AWS (opens in a new tab) · Databricks Exam Guide Databricks Certified Generative AI Engineer Associate (opens in a new tab) · Register and serve an OSS embedding model | Databricks on AWS (opens in a new tab) · Query an embedding model | Databricks on AWS (opens in a new tab) · Use a custom embedding model with AI Search | Databricks on AWS (opens in a new tab)

Select a model from a model hub or marketplace for a task based on model metadata/model cards

Start here

THINK FIRST

Given a task, how would you narrow a model hub down to one model you can safely deploy and pay for?

Why it matters: Questions describe a task and its constraints, then ask which model origin, license or serving mode fits.

Where a model comes from shapes how you serve it. Hosted base models are ready per token, and outside models arrive through External models. Provisioned throughput supports fine-tuned and custom-weight models, and Hugging Face models can also be served as MLflow custom models.

Key terms

Term	Meaning
Pay-per-token	Pricing for Databricks-hosted base models that are ready to use right away.
Provisioned throughput	A serving mode for production work needing high throughput, guarantees, fine-tuned models or extra security.
External models	The way to reach foundation models hosted outside Databricks, governed centrally from Databricks.
AI Playground	A chat-like space in the workspace to test, prompt and compare LLMs.

Where models come from

Find base models in Marketplace or bring them from Hugging Face. Base models are available through Databricks Marketplace. You can also pull one from Hugging Face or elsewhere and register it in Unity Catalog, which works for fine-tuned variants too.

Read the license

The `e5-small-v2` embedding model is released under the MIT license. It is published on Hugging Face as `intfloat/e5-small-v2`.

Ready-to-use models

Hosted base models start immediately with pay-per-token pricing. Examples include Mistral-7B, GTE-Large and Meta-Llama-3.3-70B-Instruct. Find them under AI Gateway in the sidebar, on the Models tab. System-provided services in `system.ai` work for every account user by default.

EXAM TRAP

System-provided models in `system.ai` need no extra setup. All account users can query them by default.

Matching the serving mode

Use provisioned throughput for production guarantees. Provisioned throughput also serves fine-tuned variants and fully custom weights built on a supported architecture, such as CodeLlama. Models hosted elsewhere, such as GPT-4, come in through External models.

Which option fits which kind of model

Feature	Pay-per-token	Provisioned throughput	External models
Model	Databricks-hosted base models	Base, fine-tuned or custom-weight models	Models hosted outside Databricks
Fits	Immediate use	Production needing guarantees or security	Central governance of outside providers

EXAM TRAP

A model fine-tuned on proprietary data does not run as a pay-per-token base model. Serve it with provisioned throughput.

Commitment options

Provisioned throughput comes on demand or reserved. On-demand capacity has no term commitment. Reserved capacity is held for a fixed **1**- or **3**-month term.

Serving a hub model yourself

Package a Hugging Face model as an MLflow custom model. Download it from the Hugging Face Hub, register it in the MLflow Model Registry and start a Model Serving endpoint. Custom models can live in Unity Catalog or the workspace model registry.

Compare before you commit

Test candidates side by side, then swap freely. Use AI Playground to prompt and compare LLMs. Foundation Model APIs let you swap a production model for a better one. You can also replace a proprietary model with an open one for cost and performance.

Key takeaways

- Hosted base models like Mistral-7B are ready now with pay-per-token pricing.
- Provisioned throughput supports fine-tuned and custom-weight models.
- Reach models hosted outside Databricks, such as GPT-4, through External models.
- Check a hub model's license, such as MIT for `e5-small-v2`, before using it.

Sources: Deploy models using Model Serving | Databricks on AWS (opens in a new tab) · Databricks Foundation Model APIs | Databricks on AWS (opens in a new tab) · Register and serve an OSS embedding model | Databricks on AWS (opens in a new tab) · Get started querying LLMs on Databricks | Databricks on AWS (opens in a new tab)

Select the best model for a given task based on common metrics generated in experiments

Start here

THINK FIRST

When two LLMs score similarly in your experiments, which metrics and serving choices should decide between them?

Why it matters: Questions give experiment or serving metrics and ask which model or serving option best fits the task.

Separate what the model controls from what the serving system controls. Quality comes from the model, while latency and throughput come from serving. Compare models on quality, then size serving to meet latency and load goals.

Key terms

Term	Meaning
P90 latency	The time within which 90% of requests finish; a P90 of 30 ms means 90% took 30 ms or less.
Throughput	Requests or queries per second; it falls as latency rises and grows with concurrency.
Queue time	Time requests wait before running; high values mean the endpoint is under-provisioned.
Priority mode	Pay-per-token requests sent with service_tier set to priority, admitted ahead of standard traffic.

Model versus serving

Quality belongs to the model; latency and throughput to serving. Foundation Model APIs make it easy to compare LLMs for a use case or swap in a better one. You can also replace proprietary models with open ones to balance cost and performance.

EXAM TRAP

A slow response does not prove the model is worse; latency and throughput depend on the serving system.

Reading latency metrics

The Latency metric reports P50 and P99 round trips. It measures time inside Databricks and leaves out authentication and rate limiting. The error rate metric counts 4xx and 5xx responses per second, averaged over each minute.

Throughput, concurrency and queues

Find the concurrency ratio that maximizes RPS without queuing. Throughput rises with concurrency and falls as latency grows. Queuing is on by default, so extra requests wait and their latency grows. Rising queue time means you should widen the compute scale-out range.

EXAM TRAP

Elevated queue time is not a model quality problem; it signals an under-provisioned endpoint.

Picking a serving mode

Match the serving mode to the workload's stage and needs. Priority mode needs no commitment. Reserved provisioned throughput locks capacity for **1 or 3 months**, while on-demand has no term.

Foundation Model APIs serving modes compared

Mode	Recommended for	Key detail
Pay-per-token	Getting started with Foundation Model APIs	The easiest way to begin
Priority pay-per-token	Latency-sensitive production under load	Opt-in per request, higher per-token rate
Provisioned throughput	High throughput, guarantees, fine-tuned models, security	On-demand has no term; reserved is 1 or 3 months

Sizing CPU endpoints

A Small CPU endpoint creates four model instances. Endpoint size sets how many instances of your model run on the endpoint, so check it when you plan for load.

Key takeaways

- Judge models on result quality; latency and throughput are serving traits.
- A P90 of 30 ms means 90% of requests finished in 30 ms or less.
- Provisioned throughput suits high-throughput or fine-tuned production workloads and offers HIPAA-certified endpoints.
- Widen the scale-out range when queue time climbs.

Sources: Load testing for serving endpoints | Databricks on AWS (opens in a new tab) · Databricks Foundation Model APIs | Databricks on AWS (opens in a new tab) · Track and export serving endpoint health metrics to Prometheus and Datadog | Databricks on AWS (opens in a new tab)

Utilize MLflow and Agent Framework for developing agentic systems

Start here

THINK FIRST

How do traces, scorers and judges fit together when you build, test and monitor an agent with MLflow?

Why it matters: Questions describe an agent development task and ask which MLflow object or call handles it.

A trace records one agent run as a tree of spans. Scorers read traces and return feedback, often by calling a judge. The same scorers power offline evaluation runs and live production monitoring.

Key terms

Term	Meaning
Span	One operation inside a trace. The root span is the whole request; child spans are its sub-steps.
Scorer	A function that reads a trace and returns a Feedback assessment.
Judge	An evaluator of text, such as `is_correct`, that cannot read a trace on its own.
Expectation	A domain expert's ground truth: the correct output for a given input.
Session	A tag, `mlflow.trace.session`, that groups the traces of one multi-turn conversation.

What a trace holds

A trace is one complete agent run, stored as a tree of spans. It keeps the top-level inputs and outputs and each LLM call, retriever lookup and tool call. It also records latency, token usage, cost and errors. Set `mlflow.trace.session` on each trace to group a multi-turn conversation.

Capturing traces

For LangGraph, `mlflow.langchain.autolog()` captures traces. LangGraph relies on LangChain's autolog. Mark the retrieval step with `span_type="RETRIEVER"` so retrieval-specific judges can run.

Example: Turns on automatic tracing for a LangGraph agent through LangChain autologging.

```
mlflow.langchain.autolog()
```

Scorers versus judges

Scorers pull fields from a trace and pass them to a judge. A judge such as `mlflow.genai.judges.is_correct` evaluates text but cannot read a trace. If you list a built-in judge directly in `scorers`, MLflow wraps it in a scorer for you.

Example: Passes the built-in Correctness judge straight to an evaluation, where MLflow wraps it as a scorer.

```
scorers=[Correctness()]
```

EXAM TRAP

A judge alone cannot evaluate a trace. A scorer must first extract fields such as the request, response or retrieved context.

Evaluating offline

`mlflow.genai.evaluate()` **takes a dataset and a list of scorers.** It runs the agent on every input and applies each scorer to every trace. Aggregated pass rates and metrics are stored next to the traces. Expectations are optional, but they allow the most precise scoring, such as Correctness against a known-good answer.

EXAM TRAP

You do not need expectations to get value from MLflow. Feedback is the main signal, and expectations add precision when you have them.

Monitoring in production

The same scorer works offline and on live traffic. Production monitoring runs attached scorers on incoming traces on a rolling basis. It writes the feedback back to the same trace.

Managed MLflow on Databricks

Databricks offers the same MLflow product, fully managed. You keep the standard MLflow SDK and OpenTelemetry, and you never run a tracking server or maintain versions. Traces are stored as Unity Catalog Delta tables, with governed access, SQL queries and no per-experiment storage cap.

Key takeaways

- A trace is a tree of spans: the root is the request, and children are its sub-steps.
- Scorers adapt traces for judges; judges alone cannot read a trace.
- One scorer serves both `mlflow.genai.evaluate()` and production monitoring.
- Group multi-turn traces with the `mlflow.trace.session` tag.

Sources: Core concepts for agent observability | Databricks on AWS (opens in a new tab) · Tracing overview | Databricks on AWS (opens in a new tab) · Tutorial: Evaluate and improve an agent | Databricks on AWS (opens in a new tab) · What is agent observability and quality? | Databricks on AWS (opens in a new tab)

Compare the evaluation and monitoring phases of the Gen AI application life cycle

Start here

THINK FIRST

How does checking a Gen AI app before release differ from watching its quality once real users arrive?

Why it matters: Questions describe a stage of the Gen AI life cycle and ask which evaluation or monitoring practice fits it.

Evaluation runs during development against a curated set of queries. Monitoring runs after deployment on real production traces. Both rest on the same parts, so the same scorers can serve both phases.

Key terms

Term	Meaning
Evaluation set	A curated set of queries, ideally with expected outputs, that reflects how the app is meant to be used.
Evaluation harness	A tool that runs the app on every evaluation record and scores each output.
Scorer	An MLflow 3 check that assesses the quality of an app's traces.
LLM judge	A different LLM that reviews outputs so evaluation can scale.

Two phases, shared parts

Evaluation is in development; monitoring is in production. The phases run at different times and on different inputs, but their core components are similar. Using the same scorers in both keeps results consistent, and lessons from production feed back into evaluation.

Evaluation and monitoring compared

Feature	Evaluation	Monitoring
When	During development	After deployment to production
Inputs	A curated evaluation set	A sample of incoming production traces
How it runs	A harness runs every record through judges and metrics	Scheduled scorers run on traces automatically

EXAM TRAP

Evaluation and monitoring do not need different scorers; reusing the same ones keeps the two phases consistent.

Why Gen AI needs judges

Gen AI answers can be right in many ways. Outputs are non-deterministic, and some right answers are better than others. Human feedback gives confidence, while LLM-judged metrics let evaluation scale. A separate LLM can assess groundedness over long context, which human raters cannot do at scale.

What production must log

Monitoring needs traces from MLflow Tracing. Log inputs, outputs and intermediate steps such as document retrieval. That record lets you spot and diagnose problems early once the app is live.

Starting production monitoring

Register a scorer, then start it with a sample rate. Monitoring begins when you start the scorer with .register() and .start(). Allow **15 to 20 minutes** for the first processing. An experiment can have at most **20** monitoring scorers at once.

EXAM TRAP

Class-based Scorer subclasses cannot be registered for monitoring; only @scorer decorator scorers can be serialized.

Choosing sample rates

Sample critical scorers fully and costly ones lightly. Use sample_rate=1.0 for safety and security scorers. For expensive scorers such as complex LLM judges, a rate between **0.05 and 0.2** keeps costs down.

EXAM TRAP

Running every complex LLM judge at a sample rate of 1.0 is not the recommended default; save full sampling for critical checks.

Key takeaways

- Evaluation happens in development and monitoring in production, with similar core parts.
- Reuse the same scorers across both phases for consistent results.
- Production monitoring needs MLflow Tracing and allows at most **20** scorers per experiment.
- Only @scorer decorator scorers can be registered for monitoring.

Sources: Introduction to evaluation & monitoring RAG applications | Databricks on AWS (opens in a new tab) · What is Agent Bricks? | Databricks on AWS (opens in a new tab) · Set up production monitoring | Databricks on AWS (opens in a new tab)

Enable multi-agent systems to leverage Genie Spaces or conversational API to retrieve data

Start here

THINK FIRST

How would you let an agent in a multi-agent system pull data from Genie, and which Genie option fits?

Why it matters: Questions describe an agent's data needs and governance rules and ask which Genie MCP option or setting fits.

Agents reach Genie through MCP servers instead of writing SQL themselves. The Genie Agent MCP server queries one curated Genie Agent, while Genie One MCP runs through Unity Gateway. The choice changes governance and permissions, and Genie One answers asynchronously.

Key terms

Term	Meaning
Genie Agent	A curated set of Unity Catalog tables, views or metric views, **50** at most.
Genie Agent MCP server	A read-only server that lets an agent ask one Genie Agent plain-language questions.
Genie One MCP server	A Databricks-provided MCP, named `system.ai.genie_one_mcp`, that Unity Gateway governs.
Supervisor Agent	An agent that coordinates Genie Agents, agent endpoints, functions, MCP servers and custom agents.

Two Genie MCP options

Genie Agent MCP targets one Genie Agent; Genie One MCP uses Unity Gateway. With the Genie Agent MCP server, the agent sends a plain-language question. Genie writes and runs the SQL, enforcing Unity Catalog permissions on every request.

Genie Agent MCP and Genie One MCP compared

Feature	Genie Agent MCP server	Genie One MCP server
Unity Gateway	Bypassed; gateway tracing, grants, policies and guardrails don't apply	Governed through Unity Gateway
Access needs	Permissions on the Genie Agent and its data	EXECUTE on `system.ai.genie_one_mcp`

EXAM TRAP

Unity Gateway guardrails do not protect Genie Agent MCP calls. Those calls never pass through the gateway.

Genie Agent MCP behavior

The Genie Agent MCP server is read-only and keeps no conversation history. It queries data but never writes to tables. Because it calls Genie as a tool, no history reaches the Genie API. Its scope is the Genie Agent's curated data, not the whole workspace.

EXAM TRAP

Follow-up questions do not carry earlier context. The server sends no conversation history to the Genie API.

Setup and cost

A workspace admin must turn on Managed MCP Servers first. That switch is on the workspace Previews page. Genie Agent MCP calls are billed at Databricks SQL pricing. The older Beta endpoint, `/api/2.0/mcp/genie`, is deprecated; its end of life is **November 30, 2026**.

Genie One answers asynchronously

`genie_ask` **starts an answer and returns** `in_progress` **right away.** It also returns a `conversation_id` and `response_id` while Genie keeps working. `genie_ask` and `genie_poll_response` return truncated results to protect the context window; `genie_get_query_result` returns the full schema and rows.

EXAM TRAP

Truncated results are not the whole answer. When the agent needs every row, it calls `genie_get_query_result`.

Genie One access

Calling Genie One MCP needs EXECUTE on `system.ai.genie_one_mcp`**.** Its securable type is `mcp_service`. On-behalf-of user OAuth is the recommended authentication; service principals also work for programmatic and automated workloads.

Genie inside a supervisor

A supervisor steers away from subagents a user cannot access. If the user can reach none of its subagents, the supervisor ends the conversation. A Supervisor Agent can coordinate Genie Agents alongside endpoints, functions and MCP servers.

Key takeaways

- Genie Agent MCP calls skip Unity Gateway; access rests on Genie Agent and data permissions.
- The Genie Agent MCP server is read-only and sends no conversation history.
- Genie One MCP needs EXECUTE on `system.ai.genie_one_mcp`.
- Call `genie_get_query_result` when an agent needs complete query results.

Sources: Genie Agent MCP server | Databricks on AWS (opens in a new tab) · Genie One MCP server |

Databricks on AWS (opens in a new tab) · Use Supervisor Agent to create a coordinated multi-agent system (legacy) | Databricks on AWS (opens in a new tab)

Domain 4 | 22% of the exam | 44 min read

Assembling and Deploying Applications

Overview

This domain is about 22% of the exam, roughly 10 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Code a chain using a pyfunc model with pre- and post-processing

Start here

THINK FIRST

How do you wrap a model with pre- and post-processing code so it can be served?

Why it matters: Questions describe a model that needs extra code around it and ask how to package, log or test it.

A custom pyfunc model wraps any Python code, so input preprocessing and output post-processing travel with the model. You subclass `PythonModel`, log it with its dependencies and signature, test it offline, then register and serve it.

Key terms

Term	Meaning
pyfunc	MLflow's Python function flavor, which can deploy any piece of Python code or any Python model.
load_context	Holds anything loaded only once, so fewer artifacts load during `predict`.
predict	Holds all the logic that runs every time an input request is made.
Model signature	Required for logging models to Unity Catalog; adding one with an input example is recommended.
code_path	A `log_model` parameter that logs shared code modules for use by custom pyfunc models.

When to use a custom pyfunc

Use pyfunc when a model needs code before or after it runs. Typical cases are preprocessing inputs before `predict`, or post-processing raw outputs for consumption. Custom logging with pyfunc also

deploys arbitrary Python code alongside your model.

Writing the model class

Subclass `mlflow.pyfunc.PythonModel` and implement `predict`. Put one-time loading in `load_context`, so fewer artifacts load during `predict` and inference runs faster. `predict` holds all logic that runs on every request.

Example: A minimal custom pyfunc model whose predict method doubles its input.

```
class Model(mlflow.pyfunc.PythonModel):
    def predict(self, context, model_input):
        return model_input * 2
```

- `mlflow.pyfunc.PythonModel`: the base class every custom pyfunc model subclasses

EXAM TRAP

Loading a model file inside `predict` repeats that work on every request. One-time loading belongs in `load_context`.

Logging with dependencies

Add dependencies explicitly for a custom pyfunc model. Native flavor models capture theirs automatically. On Databricks Runtime ML, pin an exact `mlflow==` version in `pip_requirements`, because Model Serving needs `mlflow`, not `mlflow-skinny`. Use `code_path` to log shared code modules.

EXAM TRAP

Leaving only `mlflow-skinny` in `conda.yaml` breaks serving. Model Serving cannot build the container image without full `mlflow`.

Signatures and offline tests

Unity Catalog needs a model signature. Adding an input example is recommended too. Test offline with `mlflow.models.predict`; its default `env_manager="virtualenv"` rebuilds dependencies in an isolated environment. Use `validate_serving_input` to check the JSON input format.

Register and serve

Register the logged model in Unity Catalog, the recommended registry. Keep model computation short on the endpoint. Requests time out if computation takes longer than **597** seconds.

Key takeaways

- Use a custom pyfunc model when inputs need preprocessing or outputs need post-processing.
- Load once in `load_context`; run per-request logic in `predict`.
- On Databricks Runtime ML, pin an exact `mlflow==` version in `pip_requirements`.
- A signature is required to log a model to Unity Catalog.

Sources: Deploy Python code with Model Serving | Databricks on AWS (opens in a new tab) · Custom models overview | Databricks on AWS (opens in a new tab) · Pre-deployment validation for Model Serving | Databricks on AWS (opens in a new tab)

Control access to resources from model serving endpoints

Start here

THINK FIRST

Why should a team identity, not a person, own the identity behind a model serving endpoint?

Why it matters: Questions describe a new or failing serving endpoint and ask which identity, grant or permission fixes it.

Every serving endpoint records the identity that created it. That creator reaches Unity Catalog resources for the endpoint, so its grants and workspace membership decide whether the endpoint keeps working.

Key terms

Term	Meaning
Recorded creator	The identity saved when an endpoint is created; it accesses Unity Catalog for the endpoint and cannot be changed.
Service principal	The recommended endpoint creator: a long-lived identity owned by your team instead of a personal account.
Workspace-access entitlement	Needed, along with workspace membership, by both caller and creator to create or update an endpoint.
CAN MANAGE	The minimum endpoint permission needed to change who can use a serving endpoint.
CAN QUERY	An endpoint permission level you can grant to an individual user so they can query it.

The recorded creator

The creator identity is fixed when the endpoint is created. Databricks saves the calling identity as the creator, typically a service principal. That identity reaches Unity Catalog resources for the endpoint. You cannot swap it later, so choose it with care.

EXAM TRAP

You cannot reassign an endpoint to a new creator. If the creator loses its grants or leaves the workspace, you delete the endpoint and recreate it under a suitable service principal.

Choose a team service principal

Make a long-lived, team-owned service principal the creator. A personal account can be deactivated or removed from the workspace later. Once the creator is gone, updates to the endpoint fail with `PERMISSION_DENIED`.

EXAM TRAP

A caller with valid permissions cannot rescue the update. It still fails once the recorded creator has left the workspace.

Workspace membership for both

Caller and creator both need membership and workspace access. To create or update an endpoint, each must be a workspace member holding the workspace-access entitlement. On every update, Databricks checks the creator's membership and served entity grants again.

Grants for a Unity Catalog model

The creator needs USE CATALOG, USE SCHEMA and EXECUTE. These grants on the catalog, the schema and the model are validated when you create or update the endpoint. If the model declares transitive function dependencies, the creator also needs EXECUTE on those upstream functions.

Grants checked only at query time

Query-time grants are not validated upfront. An endpoint can be created without errors and still fail later. A missing query-time grant shows up as a runtime error once the endpoint serves traffic.

When missing creator grants come to light

Feature	Unity Catalog model grants	Query-time grants
When validated	At endpoint creation or update	Not validated upfront
Where a gap shows	While creating or updating the endpoint	As runtime errors while serving traffic

EXAM TRAP

A successful create does not prove every grant is in place. Grants needed only at query time are tested when requests arrive.

Inheritance and parent privileges

A grant on a container flows down to its children. A privilege on a container object applies to all its current and future child objects. Reading a table still needs USE CATALOG and USE SCHEMA on its parents, not only SELECT.

EXAM TRAP

SELECT on a table alone is not enough to read it. Without USE CATALOG or USE SCHEMA on the parent objects, the read is denied.

Who can use the endpoint

CAN MANAGE is needed to change endpoint permissions. With CAN MANAGE you can give others access. For example, you can grant a single user the CAN QUERY permission on the endpoint.

Reaching outside resources

Advanced configuration links the endpoint to other resources. Add an instance profile when the endpoint must reach AWS resources. Add environment variables to connect to other resources, or to log your feature lookup DataFrame to the inference table.

Key takeaways

- The recorded creator reaches Unity Catalog for the endpoint and can never be changed.
- Make a long-lived, team-owned service principal the creator, not a personal account.
- Serving a Unity Catalog model needs USE CATALOG, USE SCHEMA and EXECUTE for the creator.
- Query-time grants are not checked upfront, so gaps appear as runtime errors.

Sources: Create custom model serving endpoints | Databricks on AWS (opens in a new tab) · Manage model serving endpoints | Databricks on AWS (opens in a new tab) · Unity Catalog permissions model concepts | Databricks on AWS (opens in a new tab)

Code a simple chain according to requirements

Start here

THINK FIRST

How do you build, trace and package a simple chain so it meets its requirements?

Why it matters: Questions give a chain's requirements and ask which step, setting or code fits them.

A RAG chain takes a user query, retrieves relevant data and generates a grounded answer. You compose its steps in code, trace them with MLflow, and package the chain with `load_context` and `predict`.

Key terms

Term	Meaning
RAG chain	A chain that retrieves data for a user query and grounds its answer on that data.
Query understanding	Opening steps that analyze and transform the query to show intent and help retrieval.

Term	Meaning
Prompt augmentation	Joining the user's query and retrieved context in a template that tells the model how to use each.
Hybrid search	Semantic search first, then keyword search on that smaller set, with both scores combined to rank.

The steps of a RAG chain

Query in, relevant data retrieved, grounded answer out. Query understanding may come first to clarify intent. Prompt augmentation then joins the query and the context. Optional post-processing can apply business logic, add citations or refine the text with predefined rules.

Composing a chain in code

The pipe operator joins the steps into one chain. Calling `mlflow.langchain.autolog()` first enables MLflow Tracing for LangChain. Each chain call then logs nested traces to the active MLflow experiment.

Works: Turns on LangChain tracing, then builds a chain from a prompt, a model and an output parser.

```
import mlflow
mlflow.langchain.autolog()
chain = prompt_template | llm | StrOutputParser()
```

- `prompt_template | llm`: the pipe operator composes the steps into one chain

Fails: This turns LangChain auto tracing off globally instead of turning it on.

```
mlflow.langchain.autolog(disable=True)
```

Tracing on serverless and production

Serverless clusters do not turn on LangChain autologging for you. Call `mlflow.langchain.autolog()` explicitly there. For production deployments, use the `mlflow-tracing` package, which is optimized for production use.

EXAM TRAP

Do not assume tracing is on in serverless compute. Autologging stays off there until you call `mlflow.langchain.autolog()`.

Retrieval choices

Embed the query with the same model used for the chunks. How many chunks you retrieve, and how you combine them with the query, can strongly affect answer quality. Hybrid search runs semantic search first, then keyword search on that smaller set.

EXAM TRAP

Query rewriting is not a standalone change. It must go together with changes to the retrieval component.

Choosing models per step

Different LLM steps can use different models. A chain with query understanding and generation steps can pick a model for each. An expensive general-purpose model may be overkill for a task like detecting a user's intent.

EXAM TRAP

One large model for every step is not required. Simpler tasks such as intent detection may not need it.

Packaging as a pyfunc model

A custom pyfunc model needs `load_context` and `predict`. Put anything loaded one time in `load_context`, which keeps `predict` fast. `predict` holds the logic that runs on every input request.

EXAM TRAP

Do not load artifacts inside `predict`. It runs on every request, so one-time loading belongs in `load_context`.

Key takeaways

- Compose a simple chain with the pipe operator: `prompt_template | llm | StrOutputParser()`.
- Call `mlflow.langchain.autolog()` explicitly on serverless compute to trace LangChain.
- Embed the retrieval query with the same model used to embed the chunks.
- Load artifacts once in `load_context`; keep per-request logic in `predict`.

Sources: Improve RAG chain quality | Databricks on AWS (opens in a new tab) · Tracing LangChain | Databricks on AWS (opens in a new tab) · Deploy Python code with Model Serving | Databricks on AWS (opens in a new tab) · RAG chain for inference | Databricks on AWS (opens in a new tab)

Choose the basic elements needed to create a RAG application: model flavor, embedding model, retriever, dependencies, input examples, model signature

Start here

THINK FIRST

Which pieces must you choose and package before a RAG application on Databricks can embed, retrieve and serve answers?

Why it matters: Questions describe a RAG build and ask which embedding model, index, endpoint or model packaging choice fits.

When you create a retriever index, you choose whether Databricks computes the embeddings or you supply them. A Delta Sync Index stays in sync with its source table. Your own models are packaged in the MLflow format so they can be served.

Key terms

Term	Meaning
Compute embeddings	The index option where Databricks creates embeddings from a text column, instead of Use existing embeddings.
Custom model	A Python model packaged in MLflow format and registered in Unity Catalog or the workspace model registry.
Model signature	The declared input and output schema of a logged model.
Input example	A sample input logged with the model alongside its signature.

Picking the embedding model

For production, use databricks-qwen3-embedding-0-6b. On standard endpoints, Databricks recommends that foundation model with a provisioned throughput serving endpoint. In LangChain, the DatabricksEmbeddings class with its endpoint parameter connects to a Foundation Model APIs embedding model.

Serving the embedding endpoint

Turn off Scale to zero on embedding endpoints. A scaled-down endpoint can take a couple of minutes to warm up, so the first index query can time out. Use CPU endpoints only for small datasets and tests, and GPU endpoints for larger data.

> **EXAM TRAP**
> Keeping the default Scale to zero risks a timeout on the first index query, because a scaled-down endpoint can take a couple of minutes to warm up.

Separate ingest and query endpoints

model_endpoint_name_for_query splits ingest from queries. This optional parameter points queries at a second endpoint that hosts the **same** embedding model. Ingestion can then use a high-throughput endpoint while queries use a low-latency one.

Choosing the retriever index

Delta Sync follows a table; Direct Vector Access is yours to write. Creating either index needs CREATE TABLE on the target schema. On standard endpoints, the source table of a Delta Sync Index must have a change data feed.

The two index types compared

Feature	Delta Sync Index	Direct Vector Access Index
Updates	Automatic and incremental from the source table	Written by you through the REST API or Python SDK
Where vectors come from	Synced from its source table as the data changes	Vectors and metadata you read and write directly

EXAM TRAP

You cannot create a Direct Vector Access Index from the UI; it needs the REST API or Python SDK.

Precomputed embeddings

Store precomputed vectors as array[float]. You choose between letting Databricks compute embeddings and using your own. On storage-optimized endpoints, the embedding dimension must divide evenly by **16**.

Packaging a custom embedding model

Log your own embedding model as a custom MLflow pyfunc. Its signature takes a single ColSpec string as input and returns a TensorSpec, with embeddings in a NumPy array. Log it with pinned pip_requirements, the signature and an input_example.

EXAM TRAP

A custom embedding model should not declare several input columns; the signature expects one ColSpec string.

Key takeaways

- For production on standard endpoints, use databricks-qwen3-embedding-0-6b with provisioned throughput.
- Remove Scale to zero on embedding endpoints to avoid first-query timeouts.
- A custom embedding model is a custom MLflow Python model with a ColSpec string input and TensorSpec output.
- Log the model with pip_requirements, a signature and an input_example.

Sources: Create AI Search endpoints and indexes | Databricks on AWS (opens in a new tab) · Deploy models using Model Serving | Databricks on AWS (opens in a new tab) · Use a custom embedding model with AI Search | Databricks on AWS (opens in a new tab) · Query an embedding model | Databricks on AWS (opens in a new tab)

Register the model to Unity Catalog using MLflow

Start here

THINK FIRST

What does a team need before it can register, share and load a model in Unity Catalog?

Why it matters: Questions describe a team registering or using a model in Unity Catalog and ask which namespace, privilege or grant applies.

A registered model in Unity Catalog is an MLflow model that Unity Catalog treats as a type of function. It lives in a three-level namespace, holds versions, and follows the same permissions model as functions.

Key terms

Term	Meaning
Registered model	An MLflow model stored in Unity Catalog and implemented there as a kind of function.
Model version	One entry inside a model container; MLflow stores each training run's artifacts and metadata as one.
Three-level namespace	The `catalog.schema.object` naming that models share with tables, views and functions.
EXECUTE	The model privilege needed to load it, view all version metadata and download model files.

Models are functions

Unity Catalog implements registered models as functions. That is why models share the functions permissions model. A model sits at `catalog.schema.object` and acts as a container for many versions.

Privileges to create and add versions

Creating a model needs CREATE MODEL on the schema or catalog. Granted on a catalog, CREATE MODEL lets you create models in any of its schemas. Workspace users already have it on the default schema of the workspace catalog.

Model privileges and what each allows

Feature	CREATE MODEL VERSION	APPLY TAG	EXECUTE
What it allows	Registering new versions only	Adding and editing tags	Loading the model and downloading its files
What it covers	New versions of the model	The model and its versions	Metadata of every version, plus model files

Granting access to a model

Use GRANT ON FUNCTION to grant privileges on a model. Because a model is a function, the grant names FUNCTION and the model's full three-level name.

Works: Grants a group the EXECUTE privilege on a registered model.

```
GRANT EXECUTE ON FUNCTION prod.ml_team.iris_model
  TO my_group;
```

Fails: Privileges on a registered model are granted with GRANT ON FUNCTION, not with a MODEL securable.

```
GRANT EXECUTE ON MODEL prod.ml_team.iris_model
  TO my_group;
```

EXAM TRAP

Writing the grant against a MODEL object is the common slip. A registered model is a function, so the grant must say ON FUNCTION.

Loading a model

Loading needs USE CATALOG, USE SCHEMA and EXECUTE. The usage privileges go on the parent catalog and schema; EXECUTE goes on the model. EXECUTE on a schema or catalog covers every current and future model inside it.

EXAM TRAP

EXECUTE on the model alone is not enough. The user also needs USE CATALOG and USE SCHEMA on the parents.

Ownership and lineage

Only a metastore admin can transfer model ownership freely. This rule prevents privilege escalation. Unity Catalog also tracks lineage automatically, from source data through models to services and dashboards.

Key takeaways

- Registered models are functions in Unity Catalog, so grants use GRANT ON FUNCTION.
- CREATE MODEL on the parent schema or catalog is required to create a model.
- Loading a model needs USE CATALOG, USE SCHEMA and EXECUTE.
- CREATE MODEL VERSION adds versions without letting the user execute or tag the model.

Sources: Unity Catalog securable objects reference | Databricks on AWS (opens in a new tab) · What is Unity Catalog? | Databricks on AWS (opens in a new tab) · Manage privileges in Unity Catalog | Databricks on AWS (opens in a new tab)

Create and query a Vector Search index

Start here

THINK FIRST

When would you build a Delta Sync Index instead of a Direct Vector Access Index, and how would you then query it?

Why it matters: Questions describe a retrieval need and ask which index type, sync mode, privilege or query option fits it.

A Vector Search index either follows a source table on its own or takes vectors you write directly. You then pick how it syncs and query it through a short list of interfaces.

Key terms

Term	Meaning
Delta Sync Index	An index that updates itself incrementally as its source table changes.
Direct Vector Access Index	An index whose vectors and metadata you read and write yourself through the REST API or Python SDK.
Sync mode	How a Delta Sync Index picks up changes: Continuous or Triggered.
ann	Approximate nearest neighbor, the default query type.

Two index types

Delta Sync follows a table; Direct Vector Access is yours to update. Choose Delta Sync when your data already lives in a table that changes over time. Choose Direct Vector Access when your own code must write vectors.

Delta Sync Index and Direct Vector Access Index compared

Feature	Delta Sync Index	Direct Vector Access Index
Who updates it	Syncs from its source table incrementally	You, through the REST API or Python SDK
Use in the UI	A Triggered sync can be started from the UI	Cannot be created in the UI; use the REST API or SDK

EXAM TRAP

A Direct Vector Access Index cannot be created in the UI, and it never updates itself from a table.

Source table rules

On standard endpoints, the source table needs a change data feed. Schema changes to the source table, such as new or modified columns, require you to rebuild the index. By default every column is synced; columns_to_sync picks a subset, but the primary key and embedding columns always stay.

EXAM TRAP

Adding a column to the source table does not flow into the index; the index must be rebuilt.

Precomputed embeddings

The embedding column must be array[float]. When you use existing embeddings, you name the column that holds them and give the embedding dimension.

Choosing a sync mode

Continuous is fast but costly; Triggered syncs on demand. Continuous runs a compute cluster for its streaming pipeline, which is why it costs more. Storage-optimized endpoints support only **Triggered** mode.

Continuous and Triggered sync modes compared

Feature	Continuous	Triggered
Freshness	In sync within seconds	Updated when you start a sync
How it runs	Automatic, on a streaming pipeline	Started from the UI, Python SDK or REST API

EXAM TRAP

Continuous mode is not available on storage-optimized endpoints, which allow Triggered sync only.

Privileges to create and query

Creating needs CREATE TABLE; querying needs three grants. To create an index you need CREATE TABLE on the target schema. To query it you need USE CATALOG on the catalog, USE SCHEMA on the schema and SELECT on the index.

Querying an index

Query through the Python SDK, REST API or vector_search(). The default query type is ann. Set query_type to hybrid for keyword plus similarity search, which returns at most **200** results by default. Filters are a dictionary on standard endpoints and a SQL-like string on storage-optimized ones.

EXAM TRAP

A filter dictionary written for a standard endpoint will not suit a storage-optimized index, which expects a SQL-like filter string.

Key takeaways

- Delta Sync follows a source table; Direct Vector Access is updated by you and cannot be made in the UI.
- Continuous sync is near real time but costs more; storage-optimized endpoints allow Triggered only.
- Querying needs USE CATALOG, USE SCHEMA and SELECT on the index.
- The default query type is ann; hybrid returns up to **200** results by default.

Sources: Create AI Search endpoints and indexes | Databricks on AWS (opens in a new tab) · Query an AI Search index | Databricks on AWS (opens in a new tab)

Identify how to serve an LLM application that leverages Foundation Model APIs

Start here

THINK FIRST

How would you serve an LLM application on Foundation Model APIs, from first prototype to production?

Why it matters: Questions describe an application's stage, load and requirements and ask which serving mode or setup fits.

Foundation Model APIs let you call Databricks-hosted models without running your own deployment. You pick a mode by workload: pay-per-token to start, provisioned throughput for production, AI Functions optimized models for batch.

Key terms

Term	Meaning
Pay-per-token	The Foundation Model APIs mode recommended for getting started.
Provisioned throughput	A mode with dedicated capacity, recommended for demanding production workloads.
Model unit	The chunk in which provisioned throughput capacity is bought.
Unity Gateway	The layer every foundation model request passes through, where rate limits, budgets and guardrails apply.

Three modes, three workloads

Each Foundation Model APIs mode fits a different workload. Pay-per-token is the easiest start. Provisioned throughput suits production that needs fine-tuned models, extra security, high throughput or performance guarantees. AI Functions optimized models suit batch inference.

EXAM TRAP

Pay-per-token is the recommended starting point, not the pick for production that needs performance guarantees. That calls for provisioned throughput.

Calling the endpoint

With the OpenAI client, pass the serving endpoint name as the model. Install the `databricks-openai` package on your cluster; its OpenAI client comes with authorization already set up. In production, authenticate with machine-to-machine OAuth tokens.

Sizing provisioned throughput

Capacity comes in model units, so you buy exactly the throughput you need. These endpoints run on serverless compute and offer compliance certifications such as HIPAA. Reserved provisioned throughput sets aside capacity for a fixed term of **1** or **3** months.

EXAM TRAP

Reserved provisioned throughput is not open-ended. Its term is fixed at **1** or **3** months.

Which models to deploy

Deploy provisioned throughput from the models pre-installed in `system.ai`. Databricks recommends these Unity Catalog models, found in catalog `system`, schema `ai`. Databricks-hosted Claude models also support prompt caching through Foundation Model APIs.

Priority requests

Set `service_tier` to `"priority"` to get ahead of standard traffic. Priority is opt-in per request, needs no commitment and costs more per token. In return, latency stays more consistent under load.

Standard and priority pay-per-token requests

Feature	Standard	Priority
Admission	Best effort	Ahead of standard best-effort traffic
Per-token rate	Standard rate	Higher rate

EXAM TRAP

Priority pay-per-token is not a capacity reservation. Each request opts in on its own, with no commitment.

Governance in the request path

Every foundation model request passes through Unity Gateway. That holds for Databricks-hosted and external models alike. There you can set rate limits, budgets and guardrails, keeping cost and access in check.

Key takeaways

- Start with pay-per-token; move to provisioned throughput for demanding production workloads.
- Use AI Functions optimized models for batch inference.
- With the OpenAI client, the `model` value is the serving endpoint name.
- Production calls should authenticate with machine-to-machine OAuth tokens.

Sources: Databricks Foundation Model APIs | Databricks on AWS (opens in a new tab) · On-demand provisioned throughput Foundation Model APIs | Databricks on AWS (opens in a new tab) · Use foundation models | Databricks on AWS (opens in a new tab)

Explain the key concepts and components of Mosaic AI Vector Search

Start here

THINK FIRST

Which parts of a vector search setup would you choose differently for a small precise app versus a billion-vector archive?

Why it matters: Questions describe a retrieval need and ask which endpoint, index type or search mode of Vector Search fits it.

Databricks AI Search, formerly Vector Search, is built into the Databricks platform. An endpoint serves each index, and Unity Catalog governs the indexes. You pick the endpoint type for scale and cost, and you choose who computes the embeddings.

Key terms

Term	Meaning
Endpoint	What serves an AI Search index; you query and update it with the REST API or the SDK.
HNSW	Hierarchical Navigable Small World, the algorithm used for approximate nearest neighbor search.
Hybrid search	Similarity search and keyword search together, with results merged by Reciprocal Rank Fusion.
Direct Vector Access Index	An index you read and write directly, keeping it updated yourself through the API or SDK.

How similarity is measured

Searches use HNSW with the L2 distance metric. For cosine similarity, normalize embeddings before indexing. With normalized vectors, L2 distance ranks results in the same order as cosine similarity.

EXAM TRAP

There is no switch to pick cosine similarity at query time. You get cosine behavior only by normalizing embeddings first.

Standard or storage-optimized

Pick the endpoint type by scale, latency and cost. Storage-optimized endpoints suit **10M+** vectors when you can accept some extra latency. They cost up to **7x** less per vector.

Endpoint types compared at dimension 768

Feature	Standard	Storage-optimized
Capacity	**320 million** vectors	Over **one billion** vectors
Indexing speed	Baseline	**10-20x** faster
Query latency	Baseline	About **250 ms** more

Choosing the embedding source

Databricks computes embeddings, or you bring your own. When you create the index, either let Databricks embed a text column or point it at precomputed embeddings. A self-managed embedding index cannot later become Databricks-managed.

EXAM TRAP

Switching to managed embeddings is not a setting change. You must create a new index and recompute the embeddings.

Direct Vector Access Index

You own the updates of a direct access index. It allows direct reads and writes of vectors and metadata. Keep it current yourself through the REST API or the Python SDK.

Hybrid keyword search

Add keyword search when exact terms matter. Keyword relevance is scored with Okapi BM25, and Reciprocal Rank Fusion merges it with similarity results. This helps RAG when data holds unique terms such as SKUs or identifiers.

Governance and access control

Unity Catalog governs indexes, but not row by row. Row and column level permissions are not supported. To restrict results per user, build application-level ACLs with the filter API.

EXAM TRAP

Unity Catalog row filters will not limit what a user retrieves from an index. Use the filter API for application-level ACLs.

Key takeaways

- Vector Search uses HNSW and L2 distance; normalize embeddings to get cosine ranking.
- Standard endpoints hold **320 million** vectors; storage-optimized hold over **one billion**.
- Hybrid search merges BM25 keyword and similarity results with Reciprocal Rank Fusion.
- A self-managed embedding index cannot be converted to a Databricks-managed one.

Sources: Databricks AI Search | Databricks on AWS (opens in a new tab) · AI Search performance guide | Databricks on AWS (opens in a new tab) · Create AI Search endpoints and indexes | Databricks on AWS (opens in a new tab)

Identify batch inference workloads and apply ai_query() appropriately

Start here

THINK FIRST

When a table of millions of rows needs an LLM's judgment, how would you run that work efficiently from SQL?

Why it matters: Questions describe a large data-processing task and ask whether and how to use ai_query for batch inference.

ai_query is the general-purpose AI Function for calling a model from SQL or Python over many rows. Reach for it when no task-specific function fits, send the whole dataset at once, and let AI Functions handle scaling.

Key terms

Term	Meaning
ai_query	A general-purpose AI Function that queries a supported model with full control of model, prompt and parameters.
Task-specific AI Function	A function built and optimized for one task; the recommended starting point before ai_query.
system.ai model services	Databricks-hosted, fully managed model services that scale automatically for batch work.
failOnError	A setting that, when false, lets a job finish and return errors for failed rows only.

When to choose ai_query

Start with a task-specific function; use ai_query when none fits. ai_query gives full control over model, prompt and parameters. It can call traditional ML models, even fully custom ones, on Model Serving endpoints, and it can query an agent on Model Serving from SQL.

Where it runs

ai_query needs Databricks Runtime 15.4 LTS or above. For the best performance, use Runtime **18.2** or later. The function is not available on Classic SQL warehouses.

EXAM TRAP

A Classic SQL warehouse cannot run ai_query. Pick a compute type that supports AI Functions.

Choosing the model

Prefer system.ai model services for batch jobs. They are fully managed and scale without provisioning. Pick a model optimized for batch inference; a non-optimized one can cut throughput and slow the job. Fine-tuned foundation models are different: they need a provisioned throughput endpoint.

Two ways to serve the model behind ai_query

Feature	system.ai model services	Provisioned throughput endpoint
Setup	Fully managed, no provisioning	You create it in Model Serving
When needed	Recommended default for batch work	Required for fine-tuned foundation models
Scaling for batch	Scales automatically	AI Functions manages it, not the endpoint's compute

Submit everything at once

Send the full dataset in a single query. AI Functions handle parallelization, retries and scaling for you. Splitting rows into small manual batches can lower throughput.

EXAM TRAP

Chunking the table into small batches yourself is not safer. It can reduce throughput, because AI Functions already parallelize the work.

Handling failed rows

Set `failOnError` to false for large workloads. The job then completes and returns error messages for the failed rows. You keep the successful results and avoid reprocessing the whole dataset.

Unity Gateway routing

Calls to system.ai services go through Unity Gateway. Usage tracking, budget integration and permission checks apply to them. Guardrails, rate limits, fallbacks, external model providers, and inference or tracing tables do not.

EXAM TRAP

Routing through Unity Gateway does not bring inference tables or guardrails. Those features are not supported for routed ai_query requests.

Putting it into production

Batch pipelines can run on a schedule or as a stream. Write the pipeline in SQL or Python. Deploy it as a Lakeflow pipeline, a scheduled Databricks workflow or a Structured Streaming inference workflow.

Key takeaways

- Use ai_query when a task-specific AI Function does not meet your needs.
- Submit the full dataset in one query; manual batching can reduce throughput.
- Set `failOnError` to false so large jobs keep their successful rows.
- ai_query needs Runtime **15.4 LTS** or above and does not run on Classic SQL warehouses.

Sources: Use ai_query | Databricks on AWS (opens in a new tab) · Query agents deployed on Databricks | Databricks on AWS (opens in a new tab) · Deploy batch inference pipelines | Databricks on AWS (opens in a new tab)

Configure vector search for a particular solution based on number of embeddings, update frequency, latency, and cost requirements

Start here

THINK FIRST

How would you pick an endpoint type, sync mode and query capacity for a vector search workload?

Why it matters: Questions give a vector count, update frequency, latency need and budget, and ask which endpoint or setting fits.

Three choices shape a vector search setup: the endpoint type, the sync mode and any extra query capacity. Standard endpoints favor low latency, while storage-optimized endpoints favor scale and cost per vector. Faster index updates and reserved throughput both raise the bill.

Key terms

Term	Meaning
Vector search unit	Capacity covering up to 2 million vectors at dimension 768 on standard endpoints, or 64 million on storage-optimized.
Triggered sync	You call the API or Python SDK to update the index; the most cost-effective sync option.
Continuous sync	The index follows the source Delta table in near real time, using a streaming cluster.
Target QPS	Extra query capacity you provision on a standard endpoint, charged regardless of actual traffic.
Hybrid search	A search mode that is resource-heavy and can significantly reduce throughput.

Standard or storage-optimized

Pick standard for latency, storage-optimized for scale and cost. Standard endpoints fit when latency is critical and the index is well under **320M** vectors. Storage-optimized endpoints hold over **one billion** vectors at dimension 768.

The two endpoint types compared

Feature	Standard	Storage-optimized
Best fit	Latency is critical; index well under 320M vectors	10M+ vectors; some extra latency is acceptable
Vectors per unit at dimension 768	Up to 2 million	Up to 64 million
Query latency	The choice when latency is critical	About 250 ms of extra latency
Cost and speed	Supports high QPS (target QPS)	Up to 7x cheaper per vector; 10-20x faster indexing

EXAM TRAP

Storage-optimized is not simply a bigger standard endpoint. Its queries add about 250 ms, and it does not support high QPS.

Sizing in vector search units

A standard unit covers up to 2 million vectors at dimension 768. Higher dimensions count more: **1 million** vectors at dimension 1536 also fill one unit. On storage-optimized endpoints, one unit covers up to **64 million** vectors at dimension 768.

EXAM TRAP

Count dimensions as well as vectors. One million vectors at dimension 1536 use a whole standard unit, not half of one.

Sync mode and update frequency

Triggered sync is the most cost-effective option. You call the API or Python SDK to update the index. Continuous sync follows the source Delta table in near real time, but its streaming cluster makes it cost more. Choose triggered sync when seconds of latency are not critical.

EXAM TRAP

Continuous sync does not come free. Its streaming cluster adds cost, so use it only when near real-time updates matter.

Throughput and target QPS

High QPS is recommended above 50 QPS of sustained throughput. Setting a target QPS provisions extra capacity, and you pay for it regardless of actual query traffic. High QPS is available for standard endpoints only.

EXAM TRAP

Target QPS capacity is not billed by use. You pay for the extra capacity even when traffic is low.

Tuning latency and throughput

Hybrid search is resource-heavy and can significantly reduce throughput. Embedding size matters too. Halving embedding dimensionality, for example from 768 to 384, typically improves QPS by about **1.5x** and cuts latency by about **20%**.

EXAM TRAP

A scale-to-zero embedding endpoint is a poor fit for real-time production. Cold starts can delay responses by minutes or even cause failures.

Keeping costs down

One endpoint can serve up to 50 indexes. Combining smaller workloads on a single endpoint can lower total costs. An endpoint is charged only after an index exists, and charges stop **24 hours** after its last index is deleted.

Key takeaways

- Standard endpoints suit critical latency; storage-optimized suit **10M+** vectors at lower cost per vector.
- Triggered sync is the cheapest; continuous sync costs more for near real-time updates.
- Target QPS capacity is billed regardless of traffic and exists only on standard endpoints.
- One endpoint serves up to **50** indexes, so combine small workloads to save cost.

Sources: AI Search performance guide | Databricks on AWS (opens in a new tab) · AI Search cost management guide | Databricks on AWS (opens in a new tab) · Scale AI Search endpoint throughput with high QPS | Databricks on AWS (opens in a new tab) · Databricks AI Search | Databricks on AWS (opens in a new tab)

Configure a persistent datastore to store and retrieve intermediate memory or structured information

Start here

THINK FIRST

How would you give a Databricks agent memory that lasts across conversations while keeping each user's data safely apart?

Why it matters: Questions describe an agent that must remember things and ask which memory option, key or setting fits the need.

Agents need two kinds of memory. Short-term memory holds context inside one conversation, while agent memory keeps facts across conversations in Lakebase. Databricks runs the storage, indexing and search for you.

Key terms

Term	Meaning
Memory store	A workspace-scoped container for memory entries, backed by Lakebase that Databricks provisions for you.
Memory entry	One stored memory, identified by its actor_id, session_id and path together.
actor_id	The key that groups memories by actor, such as an end user; it is not a permission check.
Short-term memory	Context kept inside one conversation session through thread IDs and checkpointing.

Short-term versus long-term

Pick the memory type by how long the context must last. Conversation history inside a single interaction belongs in agent sessions. Facts the agent should recall in later conversations belong in agent memory, which works with agents built on **any framework**.

Short-term memory and agent memory compared

Feature	Short-term memory	Agent memory
How long it lasts	One conversation session	Across conversations
How it works	Thread IDs and checkpointing	Entries in a memory store on Lakebase
Typical tool	Agent sessions	Agent memory, currently in **Beta**

EXAM TRAP

Agent memory is not the place for turn-by-turn chat history inside one interaction; use agent sessions for that.

Creating a memory store

A memory store is scoped to the workspace. Creating a store sets up its Lakebase storage automatically, so you manage no database yourself. Choose the store's ID with care: it is fixed after creation, and only the description can change later.

EXAM TRAP

A memory store's ID is fixed once the store exists. After creation, only the description can be updated.

Keys and isolation

The store, not actor_id, is the security boundary. An entry is keyed by actor_id, session_id and path. Anyone who can reach a store can read and write every actor's entries. For strict isolation between tenants or users, give each boundary its **own store**.

EXAM TRAP

Separating users by actor_id alone does not stop one principal reading another user's memories in the same store.

Writing entries

add() refuses an existing path; put() creates or updates. Use add() when a second write to a path should fail with **ALREADY_EXISTS**. Use put() for an upsert. Because put() is not atomic, two writes to the same path at once can overwrite each other.

EXAM TRAP

put() is not atomic, so it does not guard against concurrent writers: parallel writes to one path can overwrite each other.

Searching memories

Search ranks an actor's entries by relevance. You pass a natural-language query for one actor. Matching mixes full-text BM25 with semantic matching over content and description. One search returns at most **100** entries, with no pagination.

Memory tools and trust

Set actor_id in trusted code, never by the model. When an agent gets memory tools, your application should fill actor_id from the verified end-user identity. Letting the model pick it would let it reach memories that belong to someone else.

Other ways to persist memory

Self-managed memory on Lakebase suits SQL needs. The legacy self-managed approach uses Lakebase, a managed Postgres OLTP database, when you need direct SQL access or a custom schema. The agent-langgraph-advanced template uses LangGraph checkpointing with Lakebase for both memory types.

Cost during preview

You pay only for the Lakebase instance. While agent memory is in preview, the underlying Lakebase instance that holds the entries is billed. Agent memory itself carries no extra charge.

Key takeaways

- Use agent sessions for one conversation and agent memory for facts that must last across conversations.
- An entry is unique by actor_id, session_id and path together.

- Isolate tenants with separate memory stores, because actor_id is not an access control.
- put() upserts but is not atomic; add() fails with ALREADY_EXISTS on an existing path.
- Search returns up to **100** entries ranked by relevance, without pagination.

Sources: Self-managed agent memory (Lakebase) | Databricks on AWS (opens in a new tab) · Agent memory | Databricks on AWS (opens in a new tab)

Apply CI/CD best practices such as updating a Vector Search index, promoting prompts across environments, and testing individual components of an agent

Start here

THINK FIRST

How do you ship changes to an agent's prompts and retrieval without breaking production?

Why it matters: Questions describe a change to a deployed agent and ask which practice updates, promotes or tests it safely.

Treat prompts as versioned assets that the agent loads by alias. You promote a prompt by moving an alias, not by redeploying code. Test each piece on its own: prompts on fixed data, the search endpoint under load.

Key terms

Term	Meaning
Prompt Registry	The MLflow store where prompts are registered, versioned and loaded.
Alias	A named pointer, such as production, that the agent loads instead of a fixed version.
LoggedModel	The record `mlflow.set_active_model()` creates, linking code, settings and prompts.
Load test	A test that simulates real traffic levels to size an AI Search endpoint.

Load prompts by alias

Deployed agents should load prompts by alias, not a hard-coded version. Then a prompt can change without a redeploy. The MLflow client caches templates, so loading from the registry adds no latency.

EXAM TRAP

Hard-coding a prompt version means every prompt change needs a redeploy. Loading by alias avoids that.

Register and reference prompts

Create prompts with `mlflow.genai.register_prompt()`**.** Template variables use double braces, like `{{variable}}`. Reference an aliased prompt with `prompts:/{catalog}.{schema}.{prompt_name}@{alias}`. The schema needs CREATE FUNCTION, EXECUTE and MANAGE permissions.

Promote across environments

Promote a prompt by pointing its alias at a newer version. Aliases let you iterate in development and move a prompt through environments with no change to application code. To update production, reassign the production alias; nothing gets redeployed.

Track lineage

`mlflow.set_active_model()` **links prompt versions to app versions.** MLflow builds that lineage automatically for registry prompts. The LoggedModel holds no application code; it points to a Git commit, configuration parameters and the prompts used.

> **EXAM TRAP**
> The LoggedModel is not a code archive. It only links to code kept elsewhere, such as a Git commit.

Test prompts fairly

Evaluate every prompt version on the same dataset. A shared dataset makes the comparison fair. Keep evaluating after deployment as well, so you catch quality drops in production.

Size the search endpoint

Load-test an AI Search endpoint before production. The test runs the endpoint under several traffic levels that mimic real use, so you can size it for production needs.

Production identity

Use a service principal with an OAuth token in production. An agent deployed with Custom Agents needs manual authentication and overridden security environment variables to reach the prompt registry.

Key takeaways

- Load prompts by alias so prompt updates need no redeploy.
- Promote or update a prompt by reassigning its alias to a newer version.
- Compare prompt versions on the same evaluation dataset, and keep evaluating after deployment.
- Load-test AI Search endpoints to size them for production traffic.

Sources: Use prompts in deployed applications | Databricks on AWS (opens in a new tab) · Evaluate and compare prompt versions | Databricks on AWS (opens in a new tab) · AI Search performance guide | Databricks on AWS (opens in a new tab) · Track prompt versions alongside application versions | Databricks on AWS (opens in a new tab)

Integrate managed, external, and custom MCP servers based on a given application requirements

Start here

THINK FIRST

How do you decide between a managed, an external and a custom MCP server for an agent?

Why it matters: Questions describe an application's tools and access needs and ask which MCP server type or setup fits.

Managed MCP servers expose Databricks resources at workspace endpoints. External servers reach third-party services through Unity Catalog connections. Custom servers run your own tools on Databricks Apps.

Key terms

Term	Meaning
Workspace MCP endpoint	A managed MCP path in the workspace, such as one for a Genie space or for Unity Catalog functions.
Managed OAuth	Databricks manages the OAuth credentials for a supported provider's published MCP server.
Unity Catalog HTTP connection	The Unity Catalog object that holds an external server's connection and its auth type.
Databricks Apps	The service that hosts a custom MCP server, giving it compute and an HTTPS URL.

Managed servers need a preview switch

Managed MCP Servers must be switched on by a workspace admin first. The switch is on the workspace Previews page. A Genie space and Unity Catalog functions each have their own workspace endpoint path.

Two workspace MCP endpoint paths

Feature	Genie space	Unity Catalog functions
Endpoint path	`/api/2.0/mcp/genie/{genie_space_id}`	`/api/2.0/mcp/functions/{catalog}/{schema}/{function_name}`
Turned on by	A workspace admin enabling Managed MCP Servers	A workspace admin enabling Managed MCP Servers

EXAM TRAP

The Beta MCPs `system.ai.web_search`, `system.ai.dbsql` and `system.ai.sandbox` use a different switch. An account admin enables the Enhanced Unity Gateway preview from the account console Previews page.

Discover tools before calling

Call `list_tools()` to learn tool names and input schemas. Do this before calling any MCP tool. Databricks-provided and registered MCPs sit under the workspace's `ai-gateway/mcp-services` path, named by catalog, schema and service.

External servers with managed OAuth

Databricks can manage OAuth for Glean, GitHub, Atlassian and Slack servers. Register the provider's published MCP server as an MCP Service. Managed OAuth is the recommended choice for development and testing. It needs a Unity Catalog HTTP connection whose auth type is OAuth User to Machine Per User.

Custom servers on Databricks Apps

Databricks Apps hosts your own MCP server with compute and an HTTPS URL. Start the app name with `mcp-` so AI Playground recognizes it. For the Starter and OpenAPI templates, connect at the app URL plus `/mcp`. The OpenAPI template needs an OpenAPI 3.x spec saved as `spec.json` in a volume.

Bringing an existing server

An existing MCP server on Databricks Apps must use Streamable HTTP. It listens on `0.0.0.0` at the port in `DATABRICKS_APP_PORT`, which defaults to **8000**.

Calling an app-hosted server

A caller needs the CAN USE permission on the app plus an OAuth token. Personal access tokens are not supported for MCP servers hosted on Databricks Apps.

EXAM TRAP

A personal access token will not authenticate to an app-hosted MCP server. The caller needs an OAuth token plus CAN USE on the app.

Key takeaways

- A workspace admin enables Managed MCP Servers on the Previews page before workspace endpoints work.
- Run `list_tools()` before calling an MCP tool.
- Custom MCP servers run on Databricks Apps; start the name with `mcp-` and use `/mcp` for template apps.
- App-hosted servers need CAN USE plus an OAuth token; personal access tokens fail.

Sources: Call MCP tools from a Python agent | Databricks on AWS (opens in a new tab) · Managed OAuth providers | Databricks on AWS (opens in a new tab) · Host an MCP server on Databricks Apps | Databricks on

AWS (opens in a new tab)

Apply prompt version control and manage prompt lifecycle

Start here

THINK FIRST

How would you version, test and roll out a prompt change while keeping a full history?

Why it matters: Questions describe a prompt change and ask which registry action, function or practice handles it.

The Prompt Registry versions prompts much as Git versions code. Versions never change once created, and aliases point at the version an app should use. Moving an alias releases a new version, and history makes rollback possible.

Key terms

Term	Meaning
Prompt	A named entity in Unity Catalog, managed in the Prompt Registry.
Version	An immutable snapshot of a prompt, with an auto-incrementing number.
Alias	A mutable pointer to one specific prompt version.
Prompt URI	The `prompts:/` address an app uses to load a prompt by alias.

Versions never change

Prompt versions are immutable once created. To edit a prompt, you create a new version. This Git-like model keeps complete history, with commit messages and rollbacks.

EXAM TRAP

You cannot edit a saved version in place. Every change becomes a new version, and that history is what makes rollback possible.

Core registry functions

`register_prompt()` **creates prompts or versions;** `load_prompt()` **retrieves them.** Registering under an existing name adds a version rather than a new prompt. Removing an alias never deletes versions.

Prompt lifecycle tasks and their functions

Task	Function	What happens
Create a prompt or add a version	`register_prompt()`	An existing name gets a new version
Retrieve a prompt	`load_prompt()`	Returns a specific version or alias

Task	Function	What happens
Remove an alias	`delete_prompt_alias()`	The alias goes; the versions remain

Release with aliases

Deployed agents load prompts by alias rather than a hard-coded version. Use the URI `prompts:/{catalog}.{schema}.{prompt_name}@{alias}`. To update production, point the production alias at a newer version; application code stays as it is and is not redeployed. Cached templates mean no added latency.

Comparing versions properly

The Compare button shows only a text diff between templates. To learn which version produces better outputs, evaluate both with the same dataset and judges. Then compare the evaluation runs.

> **EXAM TRAP**
> A text diff does not tell you which prompt performs better. That takes evaluations of both versions with the same dataset and judges.

Template syntax

Registry templates use double braces, such as `{{name}}`. That applies to simple prompts and to conversations. For LangChain or LlamaIndex, which expect single braces, convert with `to_single_brace_format()`.

Access and lineage

Prompts need CREATE FUNCTION, EXECUTE and MANAGE on a schema. Those Unity Catalog schema permissions let you view or create prompts. With `mlflow.set_active_model()`, MLflow links prompt versions to application versions automatically.

Key takeaways

- Versions are immutable; edit a prompt by registering a new version.
- Aliases are mutable pointers; move one to release or update a prompt.
- Deleting an alias leaves every prompt version in place.
- Judge output quality with evaluations on the same dataset, not with the text diff.

Sources: Prompt Registry | Databricks on AWS (opens in a new tab) · Use prompts in deployed applications | Databricks on AWS (opens in a new tab) · Create and edit prompts | Databricks on AWS (opens in a new tab) · Track prompt versions alongside application versions | Databricks on AWS (opens in a new tab)

Develop an appropriate interactive user facing interface for an agent usage scenario (Apps, Slack, Teams, etc.)

Start here

THINK FIRST

What would you build so users can chat with your agent safely, and where would you host it?

Why it matters: Questions describe who will use an agent and how, then ask which hosting, interface or authentication choice fits.

Databricks recommends building agents on Databricks Apps instead of Model Serving. An app runs the agent server, can include a ready chat UI, and keeps credentials on the server rather than in the browser.

Key terms

Term	Meaning
MLflow AgentServer	An async FastAPI server in the agent template that exposes `/invocations` for querying the agent.
ResponsesAgent	An MLflow interface that makes an agent work with AI Playground, Agent Evaluation and Apps deployment.
Agent runtime	The layer that hosts the agent server on managed compute and takes care of identity and scaling.
CAN_USE	The permission an orchestrator app needs on another app it calls, along with OAuth.

Why host on Databricks Apps

Apps are the recommended place to author agents. On Apps you can use any framework or stack and add custom routes and middleware. You can also set up how users and the agent authenticate to tools and LLM endpoints.

A chat UI out of the box

Conversational templates ship with a chat interface. The UI already supports streaming, authentication and persistent history. After creating the app from a template in the workspace UI, click the app URL to open the chat.

How the code differs

Apps serve decorated functions, not a class. On Apps, the AgentServer serves module-level functions marked `@invoke()` and `@stream()`. Model Serving instead expects a ResponsesAgent class whose methods are `predict()` and `predict_stream()`.

Agent code on Databricks Apps and Model Serving

Feature	Databricks Apps	Model Serving
Agent shape	Module-level functions	Class-based ResponsesAgent

Feature	Databricks Apps	Model Serving
Entry points	`@invoke()` and `@stream()`	`predict()` and `predict_stream()`

EXAM TRAP

The `predict()` method is the Model Serving pattern. On Databricks Apps, the AgentServer looks for functions decorated with `@invoke()` and `@stream()`.

Keep secrets on the server

Let the app backend call the agent endpoint. The backend keeps tokens server-side, signs users in with corporate identity and enforces per-user permissions. Putting PATs or API keys in browser code breaks those security requirements.

EXAM TRAP

A browser page calling the endpoint with an embedded API key is not acceptable. Tokens belong in the app backend, never in the browser.

Apps calling other apps

An orchestrator app needs OAuth and CAN_USE on its target. When one Databricks app calls another, grant the CAN_USE permission on the target app and authenticate with OAuth.

Run it locally, then deploy

`start-app` **runs the server and chat UI locally.** Locally, the UI opens on port **8000**. When you deploy with bundles, `bundle deploy` only uploads files; run `bundle run` to start the app. For new agents, `agentbricks deploy` provides a stable, authenticated endpoint.

EXAM TRAP

A finished `bundle deploy` does not mean the app is live. It uploads files only, and `bundle run` starts the app.

Key takeaways

- Databricks recommends authoring agents on Databricks Apps over Model Serving.
- Conversational templates include a chat UI with streaming, authentication and history.
- Keep tokens in the app backend; never embed PATs or API keys in the browser.
- Wrapping an agent with ResponsesAgent makes it work with AI Playground and Apps deployment.

Sources: Migrate an agent from Model Serving to Databricks Apps | Databricks on AWS (opens in a new tab) · Databricks Exam Guide Databricks Certified Generative AI Engineer Associate (opens in a new tab) · Get started with agents | Databricks on AWS (opens in a new tab) · Deploy agents on Databricks | Databricks on AWS (opens in a new tab) · Build a multi-agent system on Databricks Apps | Databricks on AWS (opens in a new tab)

Domain 5 | 8% of the exam | 11 min read

Governance

Overview

This domain is about 8% of the exam, roughly 4 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Use masking techniques as guard rails to meet a performance objective

Start here

THINK FIRST

How do you keep masking guard rails from slowing down queries or model calls?

Why it matters: Questions describe a masking or guardrail setup and ask which change meets a performance goal.

Masks and filters run inside every query, so their cost adds up row by row. Keep the logic simple, let the optimizer push filters down, and skip work where you can. Guardrails on model calls follow the same idea.

Key terms

Term	Meaning
Policy UDF	The function a row filter or column mask policy runs during query execution.
SecureView barrier	The protection row filters and column masks add; it can block some optimizations.
Filter pushdown	Letting a simple predicate pass the barrier so filtering stays fast.
EXCEPT clause	The policy clause that exempts users, so the policy does not apply to them at all.

Policy UDFs run per row

UDF complexity directly drives query cost. A row filter runs its UDF for every row, and a column mask runs it for every matching value. Before production, benchmark the UDF on **1 million** rows or more.

Prefer SQL UDFs

Use SQL UDFs rather than Python UDFs in ABAC policies when you can. A Python UDF must be wrapped in a SQL UDF, and the optimizer cannot inline or optimize it. Python and Scala UDFs count as NOT DETERMINISTIC unless declared DETERMINISTIC, so a mask wrapping one blocks filter pushdown by default.

EXAM TRAP

Wrapping a Python UDF in a SQL UDF does not make it fast. The optimizer still cannot inline or optimize the Python code.

What passes the barrier

Simple predicates push down; function calls and implicit casts do not. The barrier guards against side-channel leaks. That same barrier may stop partition pruning and liquid clustering, forcing full table scans.

Predicates and the SecureView barrier

Feature	Pushed down	Blocked
Predicate type	Simple equality or basic range comparison	Calls a function or adds an implicit type cast
Example	`WHERE col > 100`	`date_format(col, ...)` inside the WHERE clause
Speed	Fast	Slower

EXAM TRAP

A policy whose UDF always returns true is not free. The barrier can still block partition pruning and liquid clustering.

Skip work for exempt users

The EXCEPT clause removes the policy entirely for exempt users. No UDF runs for them at all. When logic is too complex for principal clauses, identity functions such as `is_account_group_member()` inside the UDF resolve once at query analysis, not per row.

Error-safe, reusable masks

Use `try_divide`, `try_cast` and `try_to_number` in masking logic. They return NULL instead of throwing, so the optimizer can rearrange and fold expressions. Reuse one masking function across columns: the same UDF with the same arguments counts as one effective mask.

Mask typed columns, not documents

Materialize sensitive fields into typed columns before masking them. A `regexp_replace` mask over serialized XML or JSON works on the whole document. A mask on a scalar column handles one small value per row instead.

Guardrails on model calls

Attach a Bedrock guardrail policy only to services that need it. Put the guardrail in the AWS Region nearest your workspace, since a cross-Region call adds latency. For cheap PII checks, Guardrails AI validators such as DetectPII score output by rules, with no LLM calls.

EXAM TRAP

A Unity Gateway Python policy cannot redact. A guardrail that masks sensitive data makes it deny the whole interaction instead.

Key takeaways

- Policy UDFs run per row or per matching value; test them on at least **1 million** rows.
- Prefer SQL UDFs; Python UDFs resist optimization and block filter pushdown by default.
- Use EXCEPT to skip the policy, and its UDF, for exempt users entirely.
- Error-safe functions like `try_cast` let the optimizer rearrange masking expressions.

Sources: Performance considerations for row filter and column mask policies | Databricks on AWS (opens in a new tab) · Tutorial: Enforce Amazon Bedrock Guardrails with a custom Python policy | Databricks on AWS (opens in a new tab) · Guardrails AI scorers | Databricks on AWS (opens in a new tab)

Select guardrail techniques to protect against malicious user inputs to a Gen AI application

Start here

THINK FIRST

Which guardrail would you put in front of a Gen AI app to stop malicious inputs, and how would you set it up?

Why it matters: Questions describe an attack or risky input and ask which guardrail technique or policy setting stops it.

Guardrails check traffic before and after the model. Unity Gateway service policies can block requests in the request path, for example with Amazon Bedrock Guardrails. Guardrails AI validators score inputs and outputs by rules, without LLM calls.

Key terms

Term	Meaning
Service policy	A Unity Gateway rule that runs on requests and responses and can block them.
ApplyGuardrail API	The Amazon Bedrock call that a Python service policy sends each message to.
DetectJailbreak	A Guardrails AI scorer that flags jailbreak or prompt injection attempts in input.

Term	Meaning
Fail closed	Deny the interaction when a check fails or returns something unexpected.

Catch prompt injection

DetectJailbreak flags jailbreak and prompt injection attempts in input. It is one of several Guardrails AI validators that MLflow can use as scorers. They evaluate by rules, with no LLM calls.

Three Guardrails AI scorers compared

Feature	DetectJailbreak	DetectPII	SecretsPresent
Checks	Input	Output	Output
Looks for	Jailbreak or prompt injection attempts	Personally identifiable information	API keys, tokens or other secrets

EXAM TRAP

DetectPII and SecretsPresent check output, not input. For a malicious user prompt, DetectJailbreak is the scorer that fits.

Configuring validator scorers

Pass validator settings, such as a threshold, as constructor keyword arguments. You can also build a scorer at runtime with `get_scorer`, giving the validator name as a string.

Blocking in the gateway with Bedrock

A Python service policy can send each message to Bedrock's ApplyGuardrail API. When Bedrock intervenes, the policy blocks the interaction. The example function fails closed: an unexpected result or a failed call denies the interaction. Service policies are in Beta, and workspace admins enable them from the Previews page.

EXAM TRAP

A Python policy cannot redact. Set the guardrail's actions to Block, because a masking guardrail ends up denying the whole interaction.

Order, mode and response

Lower-ranked policies run first on requests and last on responses. Log mode records each decision but lets traffic through, so you can review the policy first. In Enforce mode, a blocked request still returns HTTP **200**.

EXAM TRAP

A request blocked in Enforce mode does not return an error code. It comes back as a successful HTTP **200** response.

Guarding MCP tools

On an MCP Service, the policy checks tool calls before the tool runs. It sends the tool name and arguments to Bedrock as INPUT. Tool results are not checked.

Keeping guardrails fast

Attach a Bedrock guardrail policy only to services that need it. Create the guardrail in the AWS Region nearest your workspace, because cross-Region calls add network latency.

Key takeaways

- Use DetectJailbreak to flag jailbreak and prompt injection attempts in user input.
- Guardrails AI scorers evaluate by rules, without LLM calls.
- Python policies cannot redact, so set Bedrock guardrail actions to Block.
- Try a new service policy in Log mode before enforcing it.

Sources: Tutorial: Enforce Amazon Bedrock Guardrails with a custom Python policy | Databricks on AWS (opens in a new tab) · Guardrails AI scorers | Databricks on AWS (opens in a new tab)

Use legal/licensing requirements for data sources to avoid legal risk

Start here

THINK FIRST

If you build on OpenAI models through Databricks, which usage requirements become your job to meet?

Why it matters: Questions describe an application built on OpenAI models and ask which required mitigation applies to it.

OpenAI models on Databricks come with mitigation requirements for high-risk use cases, and you are responsible for implementing them. Chat apps, code generation tools and apps that accept images each have their own requirement.

Key terms

Term	Meaning
Grounded or topical	An app that limits users to its subject, so general inputs cannot produce unrestricted outputs.
Human review	A person checks generated code before it is used in production.
External users	Users outside your organization; they must be authenticated or monitored.

Who carries the duty

The customer implements the mitigation requirements. When you use OpenAI models on Databricks, meeting these requirements is your responsibility.

EXAM TRAP

Running OpenAI models on Databricks does not shift the requirements to Databricks. The customer remains responsible for them.

Requirements by app type

Each kind of app has its own mitigation. Match the application to its rule. Chat apps must stay on topic, code tools need review and limits, and image inputs must stay low risk.

Mitigations for three kinds of OpenAI applications

Feature	Chat app	Code generation	Image inputs
Main rule	Grounded or topical	Human review before production	Low risk and topical images only
Also required	No unrestricted general queries	Internal users only, or client-side misuse monitoring	Applies whenever the app accepts images

Users outside your organization

Authenticate or monitor every external user. Two-factor or multi-factor authentication is one accepted mechanism. Logging individual end user IDs or IP addresses, for visibility and remediation, is another.

EXAM TRAP

Leaving external users neither authenticated nor monitored does not meet the rule. Use multi-factor authentication, or log individual user IDs or IP addresses.

Key takeaways

- You are responsible for OpenAI's high-risk mitigations when using its models on Databricks.
- Chat apps must be grounded or topical, not open to unrestricted queries.
- Generated code needs human review, plus internal-only access or misuse monitoring.

Sources: OpenAI high risk use case mitigation requirements | Databricks on AWS (opens in a new tab)

Recommend an alternative for problematic text mitigation in a data source feeding a GenAI application

Start here

THINK FIRST

When sensitive text flows through a GenAI app, where should you remove it: before logging, after storage, or before the model sees it?

Why it matters: Questions describe PII or risky text in a GenAI data flow and ask which redaction or control fits the constraint.

Where you redact decides what is protected. Client-side redaction keeps PII from ever leaving your environment, server-side redaction cleans traces already stored, and service policies stop PII from reaching the model.

Key terms

Term	Meaning
Span processor	A function MLflow runs on every span before export, used for client-side redaction.
ai_mask	An LLM-backed function that masks PII in many formats without one pattern per format.
Service policies	Controls that keep PII away from an AI service registered in Unity Catalog.
redaction_mode	The pipeline setting that chooses to mask, hash or remove PII.

Pick the redaction point

Choose client-side or server-side by what must never happen. Go client-side when sensitive values must never be transmitted or persisted. Go server-side when traces already sit in Unity Catalog and you would rather leave the agent unchanged.

Client-side and server-side trace redaction compared

Feature	Client-side	Server-side
When it runs	Before MLflow exports the span	On OTel spans already in Unity Catalog
How	Span processors set with `mlflow.tracing.configure`	`ai_mask` in a Lakeflow pipeline
Agent code changes	Yes, in the agent	None needed

Client-side span processors

Processors edit each span in place before export. Register one or more with `mlflow.tracing.configure`, and MLflow runs them on every span. Passing an empty list clears them all and stops redaction.

Example: Clears every registered span processor, which stops client-side redaction.

```
mlflow.tracing.configure(span_processors=[])
```

- `span_processors=[]`: an empty list removes all processors

Detecting PII well

Use Microsoft Presidio for better detection than regex. Its AnalyzerEngine finds entities like email addresses, credit cards and names. Its AnonymizerEngine then rewrites them.

Server-side pipeline details

`ai_mask` **handles varied PII formats on its own.** One phone number written three ways is still recognized. For custom patterns, such as employee IDs, run `regexp_replace` before `ai_mask` in the pipeline SQL. Set `redaction_mode` to mask, hash or remove.

> **EXAM TRAP**
> Custom IDs go through `regexp_replace` first, not after. The pipeline SQL applies it before `ai_mask`.

Lock down the raw tables

Only the pipeline and needed admins read raw traces. Routine analytics should query the redacted tables. Raw OTel tables get an auto time-to-live that deletes old trace data, **90** days by default. Stored traces also inherit Unity Catalog controls such as column masking and audit logging.

Redaction is not blocking

Trace redaction changes only what the trace records. The agent still receives and returns the original text. To keep PII away from an AI service registered in Unity Catalog, or to enforce one policy across the organization, use service policies.

> **EXAM TRAP**
> Masking traces does not keep PII away from the agent, which still receives the original text. To block PII from reaching a Unity Catalog-registered AI service, use service policies.

Guardrails for OpenAI use cases

Keep chat apps grounded and review generated code. A chat app should be grounded or topical, so users cannot send general inputs that produce unrestricted outputs. Any generated code needs human review before it reaches production.

Key takeaways

- Redact client-side when PII must never leave your environment.
- Redact server-side with `ai_mask` when traces are already stored and the agent must not change.
- Trace redaction does not block PII from the model; service policies do.
- Raw OTel tables keep trace data **90** days by default.

Sources: Govern and redact traces in Unity Catalog | Databricks on AWS (opens in a new tab) · OpenAI high risk use case mitigation requirements | Databricks on AWS (opens in a new tab)

Domain 6 | 12% of the exam | 27 min read

Evaluation and Monitoring

Overview

This domain is about 12% of the exam, roughly 5 of its 45 questions.

Every point below comes from Databricks's official documentation; each section lists its sources.

Select an LLM choice (size and architecture) based on a set of quantitative evaluation metrics

Start here

THINK FIRST

How would you use evaluation metrics to choose between a larger and a smaller LLM for your application?

Why it matters: Questions give evaluation results or metric types and ask which model choice or evaluation setup they support.

Build a representative evaluation set and decide which metrics matter for the app. Score the system at the overall, component and compound levels. Accuracy, latency and cost can all matter.

Key terms

Term	Meaning
Overall metric	A score on the system's end-to-end input and output, such as answer correctness or latency.
Component metric	A score on one component's inputs and outputs, such as precision@K, nDCG or toxicity.
Compound metric	A score on how components interact, such as faithfulness.
Expectations	Optional ground truth in an evaluation record, with reserved keys used by built-in judges.
Scorer	A function that returns a value or Feedback for each record; its name, or each Feedback's name, becomes the metric name.

Decide what to measure

Choose metrics that fit the application's goals. Depending on the app, key metrics may be response accuracy, latency, cost or stakeholder ratings. GenAI answers can be right in many ways, so pair human feedback for confidence with LLM-judged metrics for scale.

Three levels of metrics

Measure the whole system, each part and how parts interact. Each level looks at the same system from a different angle, from end-to-end results down to single parts.

Overall, component and compound metrics compared

Level	What it scores	Examples
Overall	The system's overall input and output	Answer correctness, latency
Component	Each component's inputs and outputs	precision@K, nDCG, latency, toxicity
Compound	How components work together	Faithfulness to retrieved knowledge

EXAM TRAP

Faithfulness is not an overall metric; it needs the retriever's internal output as well as the chain input and output.

Build the evaluation set

Use curated queries that reflect real intended use. Keep examples challenging and diverse, and update them as usage changes. MLflow datasets hold inputs, optional expectations and lineage fields; a record can carry just **one** origin type: human, document or trace.

Scoring with expectations

Built-in judges read reserved expectation keys. The keys are guidelines, expected_facts and expected_response. Live production traffic usually has no expectations, so design reused scorers to cope when they are missing.

EXAM TRAP

A scorer that assumes expectations exist will struggle online, because production traffic usually lacks ground truth.

Naming custom metrics

The scorer's function name becomes the metric name. That holds for a primitive return or one unnamed Feedback. One scorer can return a list of Feedback objects, each with a unique name, to cover several facets. Only @scorer decorator scorers work in production monitoring.

Trying candidate models

System models in system.ai work with no setup. All account users can send requests to them by default. You can also chat with them in the AI Playground before running a full evaluation.

Key takeaways

- Pick metrics such as accuracy, latency, cost or stakeholder ratings based on the app.
- Overall metrics score the whole system; component metrics score each part; compound metrics score interactions.
- Use a challenging, diverse and current evaluation set that matches intended use.
- Class-based Scorer subclasses work only offline; production monitoring needs @scorer.

Sources: Introduction to evaluation & monitoring RAG applications | Databricks on AWS (opens in a new tab) · Evaluation dataset reference | Databricks on AWS (opens in a new tab) · Code-based scorer examples | Databricks on AWS (opens in a new tab) · Get started querying LLMs on Databricks | Databricks on AWS (opens in a new tab)

Select key metrics to monitor for a specific LLM deployment scenario

Start here

THINK FIRST

For a given LLM deployment, how do you decide which infrastructure metrics and which quality scorers to watch?

Why it matters: Questions describe a deployed LLM app and ask which metric, scorer or sampling setting to monitor it with.

Monitoring has two layers. Endpoint health metrics show how the serving infrastructure behaves, while scorers judge the quality of the answers. Pick each based on the risk that matters for the scenario.

Key terms

Term	Meaning
Endpoint health metrics	Infrastructure metrics such as latency, request rate, error rate, CPU and memory usage.
Scorer	A check that MLflow 3 runs on agent traces, attaching its result as feedback to each trace.
Code-based scorer	A deterministic scorer for exact matches, format checks and performance metrics.
Inference table	A Unity Catalog Delta table that logs online requests and responses of an endpoint.

Infrastructure metrics

Watch latency, request rate, errors, CPU and memory. These endpoint health metrics appear in the Serving UI for the last **14 days** by default. You can also stream them to observability tools in real time.

Quality metrics with judges

Built-in LLM judges cover common quality dimensions. MLflow ships judges for relevance, safety, groundedness and correctness. Use code-based scorers when a check is deterministic, such as an exact match or a format rule.

Built-in judges and code-based scorers compared

Feature	Built-in LLM judges	Code-based scorers
Typical checks	Relevance, safety, groundedness, correctness	Exact matching, format validation, performance
Behavior	An LLM judges each output	Programmatic and deterministic

Running scorers in production

Register a scorer, then start it with sampling. Production monitoring, in Beta, runs scorers on a sample of traces. Allow **15 to 20 minutes** for first results. An experiment can hold at most **20** monitoring scorers, and the same scorer can serve development and production.

EXAM TRAP

Class-based Scorer subclasses cannot be registered for monitoring; only @scorer decorator scorers serialize.

Choosing what to sample

Sample critical checks fully and costly judges lightly. Use sample_rate=1.0 for safety and security scorers, and **0.05 to 0.2** for expensive LLM judges. A filter_string in ScorerSamplingConfig narrows which traces a scorer sees, with mlflow.search_traces() syntax.

EXAM TRAP

A safety scorer should not run at a low sample rate like 0.1; critical checks call for 1.0.

Logs for audits and data

Inference tables keep requests and responses in Delta. With AI Gateway, they serve quality debugging, training datasets and compliance audits. Model Serving build logs, by contrast, are kept for up to **30 days**.

Key takeaways

- Endpoint health covers latency, request rate, error rate, CPU and memory, shown for **14 days** by default.
- Built-in judges score relevance, safety, groundedness and correctness.

- Run safety scorers at sample_rate=1.0 and costly judges at **0.05 to 0.2**.
- An experiment holds at most **20** monitoring scorers at a time.

Sources: Monitor model quality and endpoint health | Databricks on AWS (opens in a new tab) · Set up production monitoring | Databricks on AWS (opens in a new tab) · Scorers and LLM judges | Databricks on AWS (opens in a new tab)

Evaluate agent performance using MLflow scoring and tracing

Start here

THINK FIRST

How do traces and scorers together tell you whether a change made your agent better or worse?

Why it matters: Questions describe an agent and a quality concern, and ask which MLflow tracing or scoring step reveals it.

Tracing shows what the agent did at each step. Scoring runs judges over a dataset and attaches their feedback to traces. Comparing scored runs then tells you whether a change helped.

Key terms

Term	Meaning
Trace	A record of each step of an agent run: inputs, outputs, latency, token usage and cost.
LLM judge	A model-based scorer, such as Safety or RetrievalGroundedness, that rates app outputs.
Ground truth	Human-labeled correct answers. Some judges, such as answer correctness, compare outputs with them.
Evaluation dataset	A Unity Catalog table of rows to evaluate. Creating one needs CREATE TABLE on a schema.

What a trace captures

A trace records every intermediate step of an agent run. For each step it keeps what went in and came out, plus latency, token usage and cost. That map of the whole run shows how the agent behaves and where a problem starts.

Turning on tracing

For LangGraph agents, LangChain autologging enables tracing. LangGraph relies on LangChain's autolog, so one call covers it. Mark the retrieval component with `span_type="RETRIEVER"` to unlock MLflow's retrieval-specific judges.

Example: Enables automatic MLflow tracing for a LangGraph agent.

```
mlflow.langchain.autolog()
```

EXAM TRAP

Retrieval-specific judges depend on the retrieval step being marked with `span_type="RETRIEVER"`. Turning on tracing is only half of the setup.

Scoring with judges

`mlflow.genai.evaluate()` **writes one scored trace per dataset row.** Each trace carries feedback from every judge. Built-in judges include Safety, which looks for harmful or inappropriate content. RetrievalGroundedness checks whether output is grounded in the retrieved data.

Which metrics need ground truth

Some judges compare outputs with human labels; others need no labels. Cost and latency need no judge at all. They are computed deterministically from the app's outputs.

How common metrics are computed

Metric	How it is computed	Needs ground truth
Answer correctness	LLM judge	Yes
Groundedness	LLM judge	No
chunk_relevance/precision	LLM judge	No
document_recall	Deterministic	Yes

EXAM TRAP

Groundedness needs no human-labeled ground truth. Answer correctness does, because it compares outputs with those labels.

Datasets and comparisons

Compare evaluation runs to confirm a new prompt version is better. The comparison view is the main way to check that. Keeping traces in an evaluation dataset links results to it. You can then follow dataset changes and every result made with it. Creating a dataset needs CREATE TABLE permission on a Unity Catalog schema.

Reviewing traces by hand

Click Assess on a trace to record a rating, note or expectation. An expectation is the correct answer. In the Traces search box, Filter with AI turns a plain-English description into a filter. On Databricks, traces can live in Unity Catalog Delta tables with access control and direct SQL access.

Key takeaways

- Traces record inputs, outputs, latency, tokens and cost for every step.
- `mlflow.genai.evaluate()` produces one scored trace per dataset row.
- Answer correctness and document_recall need ground truth; groundedness and chunk relevance do not.
- Use the run comparison view to prove a new prompt version beats the old one.

Sources: Tracing overview | Databricks on AWS (opens in a new tab) · Tutorial: Evaluate and improve an agent | Databricks on AWS (opens in a new tab) · Assess performance: Metrics that matter | Databricks on AWS (opens in a new tab) · Observe and find issues | Databricks on AWS (opens in a new tab)

Use inference logging to assess deployed RAG application performance

Start here

THINK FIRST

Once your RAG app is in production, what would you log to find out why an answer went wrong?

Why it matters: Questions describe a deployed RAG app or agent and ask how inference logging captures, limits or exposes its traffic.

Evaluation happens during development; monitoring takes over after deployment. Inference tables write each request and response to a Unity Catalog Delta table. They can also keep MLflow traces, so you can monitor and debug what users really see.

Key terms

Term	Meaning
Inference table	A Unity Catalog Delta table that logs requests and responses from a model service.
MLflow trace	A record of an agent's steps that inference tables can store for debugging and monitoring.
request_id and invocation_id	Several calls can share a request_id; invocation_id tells each individual call apart.
logging_error_codes	A column that shows when a payload was too large to log.

Evaluation versus monitoring

Monitoring is evaluation's counterpart after deployment. You evaluate while you build and monitor once the app runs in production. Production logs must record what went in, what came out and the steps between, such as document retrieval, so you can spot and diagnose issues early.

EXAM TRAP

Logging only the final answer is not enough for a RAG app. Without the retrieval step you cannot tell whether a bad answer came from poor context.

What inference tables capture

Requests and responses land in a Unity Catalog Delta table. Use the logged data to monitor, debug and optimize the model. For agents, the tables can also keep MLflow traces, which show where a run went wrong.

Turning logging on for agents

Agents deployed with `mlflow.deploy()` log automatically. No extra step is needed for those agents. For a programmatic deployment, give the endpoint configuration an `ENABLE_MLFLOW_TRACING` environment variable with the value `True` to get inference tables with traces.

Setup rules and cost

Databricks creates the table; you cannot pick an existing one. The table appears after the endpoint's first request. Enabling it needs the MANAGE privilege on the model service, for both its creator and its modifier. Logged requests and responses are billed.

EXAM TRAP

Pointing the endpoint at a table you made yourself is not supported. Databricks always creates a new inference table for you.

Expect a delay

Logs can take up to 1 hour to show up. For Foundation Model API workloads, external models and agents, log delivery is best effort. Rows can also take up to **1 hour** after the first request to appear, so an empty table right away is normal.

Payload size limits

Oversized payloads are not logged. Check `logging_error_codes` for `MAX_REQUEST_SIZE_EXCEEDED` or `MAX_RESPONSE_SIZE_EXCEEDED` when a row looks empty. The size limit depends on which gateway logs the traffic.

Size limits for the two kinds of inference tables

Feature	Unity Gateway	AI Gateway (legacy)
Largest payload logged	**10 MiB**	**1 MiB**, traces included
Oversized payload	Not logged; error code recorded	Logged as null; error code recorded

Reading and protecting the table

Use `invocation_id` to separate calls within one request. Guardrail checks or multi-turn agent calls can share a `request_id`. Leave the table's schema and name alone and never delete it, or logging can stop or the table can become corrupted.

EXAM TRAP

Renaming the table to tidy up your catalog is risky. A renamed table can stop logging data or become corrupted.

Key takeaways

- Log inputs, outputs and intermediate steps such as retrieval to diagnose a production RAG app.
- Agents deployed with `mlflow.deploy()` get inference tables automatically.
- Unity Gateway skips payloads over **10 MiB**; AI Gateway (legacy) logs over **1 MiB** as null.
- Inference logs can take up to **1 hour** to appear, and logging is billed.

Sources: Introduction to evaluation & monitoring RAG applications | Databricks on AWS (opens in a new tab) · Log requests and responses to inference tables | Databricks on AWS (opens in a new tab) · Log requests and responses for serving endpoints (legacy) | Databricks on AWS (opens in a new tab)

Use Databricks features to control LLM costs

Start here

THINK FIRST

Which Databricks controls would you combine to keep a team's LLM spending in check without breaking their work?

Why it matters: Questions describe a spending problem and ask which budget, routing or serving feature controls the cost.

You control LLM cost three ways: cap spend with Unity Gateway budgets, steer work to cheaper models, and pick the cheapest fitting serving option. Budgets act on estimates, so they limit spend rather than stop it exactly.

Key terms

Term	Meaning
Unity Gateway budget	A budget whose resource type is Unity Gateway, with spend thresholds and actions.
Smart Routing	Picks the lowest-cost model able to handle each subagent's task.
Smart defaults	A Beta feature that suggests a different agent and model as spending grows.
Scale-to-zero	Stops all replicas of an idle endpoint, so the next request waits for startup.

Budget thresholds

Set shared and per-user thresholds on a Unity Gateway budget. Daily thresholds reset at **00:00 UTC**.

Shared and per-user thresholds compared

Feature	Shared	Per-user
Whose spend	All users in the budget scope together	Each individual user
Periods	Monthly	Daily, monthly or both

Threshold actions

Choose Send email, Block usage or Trigger webhook. Blocking stops further gateway requests until the period resets or an admin raises the threshold. Enforcement uses near real-time estimates, so requests already running can finish and push spend past the limit.

EXAM TRAP

Block usage is not an exact hard cap; in-flight requests can complete, so final spend can exceed the threshold.

Counting external providers

External model spend needs a preview and a setting. Turn on the External Model Spend in Budgets preview, then choose Include external model usage. From then on, estimated spend with external providers counts against the same thresholds.

Cheaper models by default

Smart Routing and smart defaults steer work to cheaper models. Smart Routing needs a static list of system.ai models in models.model_services, nothing else. Smart defaults need an account admin to enable the Enhanced Unity Gateway preview, and they change new launches only. With Unity Gateway, open-source models like system.ai.glm-5-3 can sharply lower per-token cost for routine coding.

EXAM TRAP

Smart defaults neither switch active sessions nor restrict model access; use budget blocking and Unity Catalog permissions for that.

Serving costs

Custom LLM serving is billed per GPU instance hour. If Foundation Model APIs already serve the model you need unchanged, use them; they are simpler and highly optimized. Scale-to-zero stops idle replicas, but the next request then waits one to several minutes.

EXAM TRAP

Custom LLM serving is not the better choice when Foundation Model APIs already serve the exact model you need.

Key takeaways

- Unity Gateway budgets offer shared monthly and per-user daily or monthly thresholds.
- Budget enforcement uses estimates, so final spend can exceed a threshold.
- Smart Routing picks the lowest-cost capable model for each subagent task.
- Prefer Foundation Model APIs over custom serving when they already serve the model unchanged.

Sources: Control spending | Databricks on AWS (opens in a new tab) · Serve custom LLMs with Custom Model Serving | Databricks on AWS (opens in a new tab)

Use inference tables and Agent Monitoring to track a live LLM endpoint

Start here

THINK FIRST

How would you capture and watch what a live LLM endpoint receives and returns?

Why it matters: Questions describe a deployed endpoint or agent and ask how to log, sample or monitor its traffic.

Inference tables log an endpoint's requests and responses into Unity Catalog. For agents, they can also hold MLflow traces. Production monitoring then runs scorers on a sample of those traces and records the results.

Key terms

Term	Meaning
Inference table	A Delta table in Unity Catalog that logs serving requests and responses.
Payload table	The raw table of agent traffic that request and assessment logs are derived from.
Production monitoring	Scheduled scorers that evaluate a sample of incoming agent traces.
Sampling rate	The fraction of requests a fast inference table logs.

What inference tables log

Inference tables keep request inputs and responses in a Delta table. The table lives in Unity Catalog. For agents, Databricks creates **three** inference tables per deployment, and they can also store MLflow traces for debugging and performance monitoring.

Turning them on for agents

Agents deployed with `mlflow.deploy()` get inference tables automatically. For programmatic deployments, set `ENABLE_MLFLOW_TRACING` to True in the endpoint configuration.

EXAM TRAP

You cannot point Unity Gateway inference tables at an existing table. Databricks creates a new one after the endpoint gets its first request.

Permissions for legacy tables

Enabling AI Gateway (legacy) inference tables needs Can Manage on the endpoint. The endpoint's creator and its modifier both also need USE CATALOG, USE SCHEMA and CREATE TABLE in Unity Catalog. Payloads over **1 MiB** are logged as null, with `logging_error_codes` filled in.

Delivery delay and sampling

Log delivery for agent and Foundation Model API endpoints is best effort. Expect logs within **1** hour of a request. Request logs and assessment logs, built from the payload table, may lag further. On CPU custom model endpoints, fast inference tables log **100%** of requests unless you lower the rate.

EXAM TRAP

Inference table logs are not real time. Delivery is best effort, with logs expected within **1** hour of a request.

Monitoring with scorers

Production monitoring runs scorers on a configurable sample of traces. Results attach to each evaluated trace as feedback. If the traces live in Unity Catalog, monitoring needs a SQL warehouse ID configured.

EXAM TRAP

Monitoring traces stored in Unity Catalog does not work until a SQL warehouse ID is configured.

Enriching and retiring logs

Join with `system.serving.served_entities` for foundation model details. That system table describes the model behind the endpoint. For agents, both the request logs and the assessment logs are deprecated, and a future release removes them.

Key takeaways

- Agents deployed with `mlflow.deploy()` get inference tables automatically.
- Expect inference table logs within **1** hour; delivery is best effort.
- Production monitoring scores a sample of traces and attaches feedback to each one.
- Traces stored in Unity Catalog need a SQL warehouse ID for monitoring.

Sources: Log requests and responses for serving endpoints (legacy) | Databricks on AWS (opens in a new tab) · Set up production monitoring | Databricks on AWS (opens in a new tab) · Log requests and responses to inference tables | Databricks on AWS (opens in a new tab)

Identify evaluation judges that require ground truth

Start here

THINK FIRST

If you have no labeled answers for your evaluation data, which built-in judges can you still run, and which must wait?

Why it matters: Questions describe the data you have and ask which built-in judge needs ground truth and which does not.

Built-in judges use Databricks-hosted LLMs to score quality. Most need only inputs and outputs. A few compare against ground truth, so they also need expectations, and those judges cannot run without labels.

Key terms

Term	Meaning
Ground truth	The known correct answer or facts that a judge compares a response against.
Expectations	The judge argument that carries ground truth, such as expected facts, for each example.
Built-in LLM judge	A predefined scorer that uses a Databricks-hosted LLM to rate a quality dimension.
Multi-turn judge	A judge that takes a whole session rather than one input and output.

Judges that need ground truth

Correctness, RetrievalSufficiency and ToolCallCorrectness need it. Each one takes expectations alongside inputs and outputs, so you need labeled data before you can use them.

Built-in judges that require ground truth

Judge	Arguments	What it checks
Correctness	inputs, outputs, expectations	Whether the response matches the provided ground truth
RetrievalSufficiency	Uses expectations	Whether the context holds everything needed for the ground truth facts
ToolCallCorrectness	Uses expectations	Whether tool calls and arguments are right for the query

EXAM TRAP

RetrievalSufficiency is not the same as RetrievalRelevance; sufficiency needs ground truth, relevance does not.

Judges that need only inputs and outputs

Relevance, safety and groundedness judges need no labels. RelevanceToQuery, RetrievalRelevance, Safety and RetrievalGroundedness run on inputs and outputs. So does ToolCallEfficiency. RetrievalGroundedness checks for hallucination against the retrieved context, not against ground truth.

EXAM TRAP

Detecting hallucination does not require labels; RetrievalGroundedness compares the response with the retrieved context.

Guideline judges

Neither guideline judge requires ground truth. Guidelines() applies one set of global rules to every row and works offline and in production monitoring. ExpectationsGuidelines() applies per-row guidelines that experts label, passed in expectations, and runs in offline evaluation only.

EXAM TRAP

ExpectationsGuidelines() cannot run in production monitoring; it is for offline evaluation only.

Multi-turn judges

Session judges take a session and need no ground truth. ConversationCompleteness, UserFrustration and KnowledgeRetention look at a whole conversation rather than a single turn.

Getting ground truth

Synthetic evaluations can supply expectations. With MLflow 3, generate_evals_df returns expectations holding expected_facts and expected_retrieved_context for each example. Note that judge alignment supports neither agent-based nor expectation-based evaluation.

Key takeaways

- Correctness, RetrievalSufficiency and ToolCallCorrectness all require ground truth.
- RetrievalGroundedness catches hallucination without ground truth.
- Guidelines() works offline and online; ExpectationsGuidelines() is offline only.
- Multi-turn judges take a session and need no ground truth.

Sources: Built-in LLM judges | Databricks on AWS (opens in a new tab) · Synthesize evaluation sets | Databricks on AWS (opens in a new tab) · Create a guidelines LLM judge | Databricks on AWS (opens in a new tab) · Align LLM judges with human feedback | Databricks on AWS (opens in a new tab)

Use AI Gateway (Inference Tables, Usage Tables, and rate limiting) to track an LLM or agent deployed via Agent Framework

Start here

THINK FIRST

How do you watch, limit and audit traffic to an LLM or agent you have deployed?

Why it matters: Questions describe a monitoring or cost-control need and ask which AI Gateway feature or setting meets it.

Inference tables log each request and response, the usage table tracks calls to each service, and rate limits cap traffic. Together they let you monitor, attribute and control a deployed model or agent.

Key terms

Term	Meaning
Inference table	A Unity Catalog Delta table that logs a model service's requests and responses.
Usage table	The `system.ai_gateway.usage` system table, which records requests to gateway services.
Rate limit	A cap on requests per minute (QPM) or tokens per minute (TPM) for a service.
Request tags	Labels sent in a request header to attribute usage to projects, teams or end users.

Inference tables log payloads

Inference tables store requests and responses in Delta tables. Use them to monitor, debug and optimize models. They are billed for the requests and responses they log. Payloads over **10 MiB** are skipped, and `logging_error_codes` flags it.

Setting up inference tables

Databricks creates the table; you cannot pick an existing one. It appears after the endpoint's first request, and rows can take up to an hour to show. You need `MANAGE` on the service, `CREATE TABLE` in the schema, `USE CATALOG` and `USE SCHEMA`.

EXAM TRAP

Do not rename, delete or change the schema of an inference table. It can stop logging data or become corrupted.

Tracking usage

`system.ai_gateway.usage` **records requests to gateway services.** It covers model services, model provider services and MCP services. Send the `Databricks-Ai-Gateway-Request-Tags` header to attribute usage to teams or projects.

EXAM TRAP

Ordinary users cannot query the usage table by default. Both account admin and metastore admin roles are needed unless admins grant access.

Rate limit types

Model services take QPM and TPM limits; MCP services take QPM only. No rate limits exist by default. A service can have up to **20** limits, with at most **5** group-specific ones.

Rate limits on model and MCP services

Feature	Model services	MCP services
Requests per minute (QPM)	Supported	Supported
Tokens per minute (TPM)	Supported	Not supported

Which limit wins

The more restrictive of a request and a token limit applies. A user-specific limit beats a group-specific one. The service limit is a global maximum that blocks all requests once exceeded. Blocked calls get HTTP **429**, so clients should retry with exponential backoff.

EXAM TRAP

A generous user limit does not bypass the service limit. When the service maximum is exceeded, every request is blocked.

Key takeaways

- Inference tables log requests and responses to Delta tables; payloads over **10 MiB** are skipped.
- Databricks creates a new inference table for you; rows can take up to an hour to appear.
- MCP services accept only QPM limits; model services accept QPM and TPM.
- Exceeding a rate limit returns HTTP **429**; clients should retry with exponential backoff.

Sources: Log requests and responses to inference tables | Databricks on AWS (opens in a new tab) · Track model usage | Databricks on AWS (opens in a new tab) · Apply rate limits to model and MCP services | Databricks on AWS (opens in a new tab)

Use Databricks custom Scorers for evaluating agents and LLMs

Start here

THINK FIRST

When should you write your own scorer instead of relying on a built-in LLM judge?

Why it matters: Questions describe an evaluation need and ask which custom scorer type or setup fits it.

Built-in judges cover common checks. When they don't fit, write a custom judge with `make_judge` or a code-based scorer in Python. The scorer type also decides whether it can run in production monitoring.

Key terms

Term	Meaning
Code-based scorer	A Python function you write to score agent output when built-in and custom LLM judges don't fit.
@scorer decorator	The recommended way to define most code-based scorers as plain functions.
Scorer class	A subclass for scorers that need state, complex initialization or Pydantic fields.
make_judge	An MLflow function that creates an LLM judge tailored to your own use case.

When to write scorer code

Code-based scorers fill the gaps LLM judges leave. Use one for a custom heuristic or programmatic metric. Use one to change how trace data maps to a built-in judge. Use one to evaluate with your own LLM rather than a Databricks-hosted judge.

Decorator or class

Start with `@scorer`; use the `Scorer` class only when you need it. The decorator suits most cases. A `Scorer` subclass fits when the scorer needs state, complex initialization or Pydantic fields.

The two ways to define a code-based scorer

Feature	@scorer function	Scorer class
Use it for	Most cases; the recommended starting point	State, complex initialization or Pydantic fields
Production monitoring	Supported	Not supported

EXAM TRAP

A `Scorer` subclass cannot run in production monitoring. For a stateful production scorer, use `@scorer` and keep the state inside the function body.

One scorer, development and production

The same code-based scorer can evaluate and monitor. `mlflow.genai.evaluate()` runs your scorers during development, and the same scorer can then monitor production. Production `@scorer` functions have to be created and registered in a Databricks notebook.

> **EXAM TRAP**
> Registering a monitoring scorer outside a Databricks notebook does not work. The monitoring service serializes the function code, and that step needs the notebook environment.

Custom LLM judges

`make_judge` **builds an LLM judge for your own criteria.** Its instructions reference template variables such as `{{ inputs }}`, `{{ outputs }}` and `{{ expectations }}`. One example judge rates an answer's markdown formatting and accuracy as high, medium or low quality.

Judges in prompt optimization

`optimize_prompts` **takes custom judges through its** `scorers` **argument.** The optimizer works with numbers, so you also pass an `aggregation` function. It turns the judge's feedback into a numeric score.

Example: Runs prompt optimization with a custom judge and a feedback-to-score function.

```
result = mlflow.genai.optimize_prompts(
    predict_fn=predict_fn,
    train_data=dataset,
    prompt_uris=[prompt.uri],
    optimizer=GepaPromptOptimizer(...),
    scorers=[markdown_output_judge],
    aggregation=feedback_to_score
)
```

- `scorers=[markdown_output_judge]`: passes the custom judge
- `aggregation=feedback_to_score`: maps the judge's feedback to a number

> **EXAM TRAP**
> The optimizer cannot use a judge's feedback as it comes. The feedback has to be mapped to a numeric score first.

Key takeaways

- Start with the `@scorer` decorator; use a `Scorer` class only for state, complex setup or Pydantic fields.
- Production monitoring runs built-in judges and `@scorer` functions, not `Scorer` subclasses.
- Define and register production `@scorer` functions from a Databricks notebook.
- For prompt optimization, map judge feedback to a numeric score with an aggregation function.

Sources: Code-based scorers | Databricks on AWS (opens in a new tab) · Optimize prompts using custom scorers | Databricks on AWS (opens in a new tab)

Incorporate SME feedback to improve agent performance

Start here

THINK FIRST

How do you turn what subject-matter experts know into better agent versions?

Why it matters: Questions describe an expert review need and ask which MLflow labeling feature or setting fits.

Experts review copies of agent traces in a labeling session and record labels on them. Ground-truth labels then become evaluation datasets, so you can test each new agent version against what experts expect.

Key terms

Term	Meaning
Labeling session	A special MLflow Run that queues traces for chosen experts to review with selected schemas.
Labeling schema	Defines what experts record; it is either an expectation type or a feedback type.
Expectation	A ground-truth label, such as expected facts, that can feed an evaluation dataset.
Review App	The app where experts define good responses and help align LLM judges.

Why experts review agents

Experts define what a high-quality response looks like. You engage them through the Review App. Their input also aligns LLM judges with your business requirements. For new human-review workflows, Databricks recommends review queues, currently in Beta.

Two kinds of labeling schema

Expectation schemas capture answers; feedback schemas capture judgments. Pick the type by what the expert gives you. Built-in schemas include `EXPECTED_FACTS`, `EXPECTED_RESPONSE` and `GUIDELINES`, or you can create custom ones.

Expectation and feedback schemas compared

Feature	Expectation	Feedback
What the expert gives	A ground truth or correct answer	A subjective assessment, rating or classification
Example	`expected_facts` for a RAG response	A rating of a response
Schema setting	`type="expectation"`	`type="feedback"`

EXAM TRAP

A rating is not ground truth. Ratings and classifications belong in feedback schemas, while correct answers belong in expectation schemas.

Running a labeling session

A labeling session works as a review queue of traces. It is a special type of MLflow Run. Traces are copied into it, so labels never change your original logged traces. Experts need `CAN_EDIT` permission on the MLflow experiment.

EXAM TRAP

Labeling does not alter the logged traces. The session works on copies, so the originals stay as they were.

Session size and tracking

Aim for 25-100 traces per session. That range helps avoid reviewer fatigue. Session names might not be unique, so store and reference each session by `session.mlflow_run_id`.

EXAM TRAP

Do not look a session up by its name. Names can repeat, so use the MLflow run ID instead.

From labels to evaluation

Labels are stored as Assessment objects on each trace. A trace with a ground-truth expectation becomes a ready-made test case. When expectations sync to an evaluation dataset, each trace's inputs act as the record key. You then test new agent versions with `mlflow.genai.evaluate()`.

Review App Chat UI limits

The Chat UI needs an agent on a Model Serving endpoint. For now, it cannot reach agents deployed on Databricks Apps. Plan the deployment target before you invite experts to chat with the agent.

Key takeaways

- Expectation schemas capture ground truth; feedback schemas capture ratings and judgments.
- Labeling sessions copy traces, so expert labels never change the originals.
- Keep sessions to **25-100** traces and track them by `session.mlflow_run_id`.
- Expectation labels feed evaluation datasets used with `mlflow.genai.evaluate()`.

Sources: Enable experts and collaborators to label traces | Databricks on AWS (opens in a new tab) · Collect feedback and build datasets | Databricks on AWS (opens in a new tab) · Create and manage labeling sessions | Databricks on AWS (opens in a new tab) · Test an agent with the Chat UI | Databricks on AWS (opens in a new tab)

MockCert

Take a Databricks Certified Generative AI Engineer Associate mock exam the way the real one feels

5 timed mock exams · 45 questions in 90 minutes each · one is free

- Free to start: study notes, explained quizzes, flashcards and the first mock exam
- Timed mock exams built to the official exam guide
- A verifiable Certificate of Assessment when you pass a premium mock

Explore Databricks Certified Generative AI Engineer Associate o…

Open it on your phone

mockcert.com/databricks-genai-engineer-associate/

Scan the code with your phone's camera.